С.М. Диго

БАЗЫ ДАННЫХ:
проектирование и использование

Рекомендовано Министерством образования и науки
Российской Федерации
в качестве учебника дня студентов
высших учебных заведений,
обучающихся по специальности
Прикладная информатика (по областям)

МОСКВА
"ФИНАНСЫ И СТАТИСТИКА"
2005

РЕЦЕНЗЕНТЫ:

кафедра «Информационные технологии»

Финансовой академии при Правительстве РФ

(заведующий кафедрой профессор,

доктор экономических наук Д.В. Чистов);

А.М. Вендров,
профессор кафедры «Прикладная информатика в экономике»
Всероссийской государственной налоговой академии,
кандидат технических наук

Диго С.М.

Базы данных: проектирование и использование: Учебник. - М.: Финансы и статистика, 2005. - 592 с: ил. ISBN 5-279-02571-2

Даны общая характеристика и классификация баз данных. Рассмотрены этапы проектирования баз данных. Особое внимание уделено методологии построения ER-моделей и использованию CASE-систем при проектировании баз данных. Описаны табличный язык запросов и язык SQL, возможности получения отчетов и создания экранных форм, способы обеспечения целостности и безопасности данных, особенности распределенных баз данных.

Для студентов вузов, обучающихся по специальностям «Прикладная информатика в экономике», «Прикладная информатика в менеджменте», а также для специалистов в области разработки экономических информационных систем.

Предисловие....8
Глава 1 ВВЕДЕНИЕ В БАНКИ ДАННЫХ....13
1.1. Понятие банка данных....13
1.2. Компоненты банка данных....19
1.2.1. Информационный компонент....20
1.2.2. Программные средства БнД....22
1.2.3. Языковые средства БнД....23
1.2.4. Технические средства БнД....28
1.2.5. Организационно-методические средства....31
1.2.6. Администраторы банка данных....32
1.2.7. Взаимодействие компонентов БнД....36
1.3. Классификация банков данных....36
1.3.1. Классификация баз данных....39
1.3.2. Классификации СУБД....46
1.3.3. Классификационные группировки, относящиеся к БнД в целом....49
1.4. Выбор СУБД....51
1.4.1. Тенденции развития СУБД....51
1.4.2. Общая характеристика проблемы выбора СУБД....53
1.4.3. Факторы влияния на выбор СУБД....54
1.4.4. Выбор СУБД....57
1.5. Уровни моделей и этапы проектирования БД....59
1.5.1. Уровни моделей....59
1.5.2. Взаимосвязь этапов проектирования БД....61
1.5.3. Факторы влияния на проектирование БД....64
На это следует обратить внимание....66
Контрольные вопросы....66
Глава 2 КОНЦЕПТУАЛЬНОЕ ПРОЕКТИРОВАНИЕ....69
2.1. Общие сведения о моделировании предметной области....69
2.1.1. Уточнение понятия концептуальной модели....69
2.1.2. Основные компоненты концептуальной модели....72
2.1.3. Требования, предъявляемые к концептуальной модели....72
2.1.4. Преимущества использования ER-моделирования....74
2.2. Описание базовой ER-модели....75
2.2.1. Понятия «объект» и «класс объектов»....76
2.2.2. Разновидности объектов....77
2.2.3. Изображение простого объекта....78
2.2.4. Описание свойств объекта. Разновидности свойств....79
2.2.5. Алгоритмические зависимости....84
2.2.6. Интегральные характеристики класса объектов....85
2.2.7. Связи между объектами....87
2.2.9. Рекомендации по построению базовой ER-модели....97
2.3. Сравнение методик построения ER-моделей....101
2.3.1. Несущественные различия в использовании условных обозначений....102
2.3.2. Различия в использовании и изобразительных средств, приводящие к изменениям в методике построения модели....106
2.3.3. Пространственное размещение элементов ER-модели....109
2.3.4. Отсутствующие возможности....109
2.3.5. Различия в классификации объектов и отношений между ними....112
2.3.6. Терминологические различия....112
2.3.7. Соглашения по именованию элементов ER-модели....113

2.3.8. Дополнительные характеристики CASE-средств 114
2.3.9. Использование графических ПП для изображения ER-моделей 119
2.4. Особенности методологии построения ER-моделей 123
2.5. Использование Design/IDEF для проектирования баз данных 127
2.5.1. Построение ER-модели при использовании Design/IDEF 127
2.5.2. Методология построения ER-модели при использовании Design/IDEF 143
2.6. Особенности моделирования в ERWin 154
2.6.1. Общие замечания 154
2.6.2. Построение логической модели 156
На это следует обратить внимание 176
Контрольные вопросы 176
Глава 3 ДАТАЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ 178
3.1. Общие сведения о даталогическом проектировании 178
3.1.1. Исходные данные для даталогического проектирования 178
3.1.2. Результат даталогического проектирования 178
3.1.3. Подход к даталогическому проектированию 179
3.1.4. Определение состава базы данных 181
3.1.5. Введение искусственных идентификаторов 183
3.1.6. Критерии оценки БД 183
3.2. Особенности даталогических моделей 191
3.2.1. Внутризаписная структура 191
3.2.2. Межзаписная структура 192
3.3. Проектирование логической структуры реляционной базы данных 195
3.3.1. Вводные положения 195
3.3.2. Алгоритм перехода от базовой ER-модели к схеме реляционной базы данных 196
3.4. Создание физической модели в ERWin 209
3.4.1. Выбор целевой СУБД 209
3.4.2. Нотации, используемые при построении физической модели 210
3.4.3. Уровни просмотра физической модели 215
3.4.4. Сравнение логической и физической моделей 216
3.4.5. Создание хранилищ данных 218
3.4.6. Переход к даталогической модели 222
На это следует обратить внимание 223
Контрольные вопросы 224
Глава 4 ЦЕЛОСТНОСТЬ БАЗЫ ДАННЫХ 226
4.1. Классификация ограничений целостности 226
4.2. ER-модели и ограничения целостности 236
4.3. Задание ограничений целостности в ERWin 237
4.3.1. Обязательный атрибут 238
4.3.2. Ограничения целостности связи 238
4.3.3. Триггер ссылочной целостности 241
На это следует обратить внимание 241
Контрольные вопросы 242
Глава 5 СОЗДАНИЕ И ВЕДЕНИЕ БАЗ ДАННЫХ 243
5.1. Описание структуры баз данных. 243
5.2. Создание БД в Microsoft Access 246
5.2.1. Создание новой таблицы путем описания ее структуры 248
5.2.2. Изменение структуры таблиц 264
5.2.3. Другие способы создания таблиц 264
5.2.4. Связывание таблиц 266
5.2.5. Просмотр связанных таблиц 268
5.2.6. Задание ограничений целостности в Access 271

5.3. Организация ввода и корректировки данных в БД 277
5.3.1. Общие сведения 277
5.3.2. Возможности ввода данных в Access 281
На это следует обратить внимание 283
Контрольные вопросы 283
Глава 6 ЯЗЫК ЗАПРОСОВ QBE 284
6.1. Общая характеристика языка QBE 284
6.2. Реализация QBE в Access 289
6.2.1. Общие сведения 289
6.2.2. Добавление таблиц в запросе 290
6.2.3. Удаление таблицы из запроса 292
6.2.4. Включение полей в запрос 292
6.2.5. Поля, выводимые в ответ 293
6.2.6. Управление выводом повторяющихся строк 294
6.2.7. Простые запросы 296
6.2.8. Сложные запросы 298
6.2.9. Просмотр ответа 299
6.2.10. Определение числа записей, выводимых в ответ 300
6.2.11. Формирование запросов к связанным таблицам 301
6.2.12. Выполнение агрегирующих операторов 308
6.2.13. Вычисляемые поля 310
6.2.14. Перекрестные запросы 312
6.2.15. Создание запроса с параметрами 317
6.2.16. Корректирующие запросы 318
6.2.17. Запрос на создание таблицы 324
6.2.18. Специальные запросы 326
6.2.19. Режим сводной таблицы и сводной диаграммы 332
На это следует обратить внимание 335
Контрольные вопросы 335
Глава 7 ЯЗЫК SQL 337
7.1. Общая характеристика SQL 337
7.2. Описание базы данных 338
7.2.1. Описание таблиц 338
7.2.2. Ограничения целостности 339
7.3. Запросы на выборку 341
7.4. Возможности корректировки хранимых данных 342
7.5. Создание представлений (VIEW) 343
7.6. Создание и использование курсоров 343
2.7. Управление транзакциями 343
7.8. Стандартный SQL-92 344
7.8.1. Создание объектов 344
7.8.2. Запросы 348
7.8.3. Создание представлений (VIEW) 360
7.8.4. Курсоры 368
7.9. MS Jet Access SQL 369
7.9.1. Оператор SELECT 370
7.9.2. Подчиненные запросы SQL 381
7.9.3. Корректирующие операторы 383
7.9.4. Запрос к серверу 386
На это следует обратить внимание 387
Контрольные вопросы 387
Глава 8 СОЗДАНИЕ ЭКРАННЫХ ФОРМ И СТРАНИЦ ДОСТУПА 390

8.1. Понятие, классификация и роль экранных форм 390
8.2. Рекомендации по созданию форм 392
8.3. Создание экранных форм в СУБД Access 395
8.3.1. Выбор способа создания формы 395
8.3.2. Создание форм с помощью Мастера 395
8.3.3. Корректировка формы в режиме Конструктор 403
8.3.4. Кнопочная форма 417
8.3.5. Возможные случаи возникновения ошибок 422
8.3.6. Открытие формы в режиме сводной таблицы или в режиме диаграммы 423
8.3.7. Создание страниц доступа 426
На это следует обратить внимание 426
Контрольные вопросы 427
Глава 9 СОЗДАНИЕ ОТЧЕТОВ 429
9.1. Общая характеристика отчетов 429
9.2. Создание отчетов в системе Access 432
9.2.1. Выбор способа создания отчета 433
9.2.2. Создание отчетов с использованием Мастера отчетов 434
9.2.3. Корректировка отчета в режиме Конструктор 440
9.2.4. Создание отчета, базирующегося на нескольких таблицах 453
9.2.5. Создание сложных отчетов 454
9.2.6. Свойства 455
9.2.7. Создание отчета анкетной формы 456
9.2.8. Совместная работа с другими приложениями MS Office 456
На это следует обратить внимание 459
Контрольные вопросы 459
Глава 10 РАСПРЕДЕЛЕННЫЕ БАНКИ ДАННЫХ 461
10.1. Основные понятия 461
10.2. Классификация РБнД 462
10.3. Транзакции 468
10.3.1. Понятие транзакции 468
10.3.2. Плоские транзакции 469
10.3.3. Контрольные точки 469
10.3.4. Многозвенные транзакции 470
10.3.5. Вложенные транзакции 470
10.4. Проблемы параллелизма и пути их решения 471
10.4.1. Параллелизм 471
10.4.2. Блокировки 472
10.4.3. Режимы доступа к информации 474
10.4.4. Уровни изоляции в SQL 475
10.4.5. Использование хранимых процедур и триггеров для контроля целостности БД 476
10.5. Тиражирование данных 477
10.5.1. Основные понятия 477
10.5.2. Преимущества и недостатки тиражирования 478
10.5.3. Виды тиражирования 478
10.6. Обеспечение целостности и безопасности данных в РБД 483
10.6.1. Особенности обеспечения целостности в РБД 483
10.6.2. Средства защиты данных 484
10.7. Работа в распределенной среде при использовании СУБД Access 490
10.7.1. Способы совместного использования данных в Access 490
10.7.2. Виды блокировок 494
10.7.3. Проекты Microsoft Access 495
10.7.4. Средства защиты Microsoft Access 498

На это следует обратить внимание.....501
Контрольные вопросы 501
ПРИЛОЖЕНИЯ 503
1. Основные понятия реляционной модели данных 503
2. Сквозной пример использования ER-моделирования для проектирования БД 504
ГЛОССАРИЙ 508
ЛИТЕРАТУРА 514
СОКРАЩЕНИЯ 516

Предисловие

В настоящее время наблюдается тенденция роста информатизации и компьютеризации общества, причем эта тенденция отмечается повсеместно - во всех странах мира, в разных областях профессиональной деятельности и в быту. Ядро любой информационной системы (ИС) составляет информационная база, преобладающей формой организации которой является база данных (БД).

Базы данных представляют собой особую организацию данных и предполагают использование специальных программно-языковых средств (систем управления базами данных - СУБД), а также специфического подхода к организации ИС. Проектирование баз данных подчиняется, с одной стороны, общим правилам проектирования информационных систем, а с другой - имеет отличительные особенности, свойственные «банковской» организации данных.

В связи с тем, что базы данных - это, по сути, сложные взаимосвязанные информационные образования, предназначенные для многоцелевого использования, проектирование баз данных можно рассматривать как более общий случай проектирования информационной базы по сравнению, например, с «позадачным» («файловым») подходом к проектированию. Поэтому все, что «наработано» в области проектирования баз данных, может быть использовано и при создании систем других типов.

Предметом курса «Базы данных» являются базы данных как ядро банка данных, а также языковые средства банка данных (БнД) и средства проектирования баз данных. Цель курса - изучение теоретических основ проектирования баз данных, а также особенностей СУБД разных классов. Несмотря на то, что в книге не рассматривается ни одна конкретная СУБД в полном объеме, предполагается, что в учебном процессе будет выбрана одна (или несколько) СУБД, на примере которой студенты приобретут навыки практической работы. Предусматривается выполнение комплекса лабораторных работ, состав которых будет зависеть от конкретной специальности, а также от технического и программного обеспечения, имеющегося в распоряжении в учебном заведении.

Учебник имеет следующую структуру: предисловие, десять глав, список литературы, список сокращений, глоссарий, приложения.

И главе 1 дано понятие банка данных, определен его состав с крат кой характеристикой каждого из компонентов. Приведены классификации БнД, языковых средств современных СУБД и пользователей банков данных. Описаны основные черты языков 4-го поколения, функции администратора БД и др. Рассмотрены также проблемы, связанные с выбором СУБД. Завершается глава общей характеристикой этапов проектирования банков данных.

Главу 2, посвященную концептуальному моделированию предметной области, следует рассматривать как центральную в учебнике. Концептуальная модель играет важнейшую роль в информационной системе:

- способствует лучшему пониманию предметной области; она должна строиться при проектировании не только банков данных, но и информационных систем других типов;
- является связующим звеном между всеми категориями специалистов, участвующих в создании и функционировании банков данных;
- служит основанием для дальнейшего проектирования баз данных;
- используется как при ручном, так и при автоматизированном проектировании.

В этой главе приведены общие сведения о концептуальном моделировании, предложена авторская методика построения концептуальной модели (базовая ER-модель), а также дан анализ других нотаций и методик, описанных в литературных источниках и используемых в наиболее известных системах автоматизации проектирования.

Тема концептуального моделирования в настоящее время достаточно широко освещена в литературе, имеются попытки стандартизации в этой области [4], динамично развивается рынок CASE-средств, сердцевиной которых является концептуальная модель. Однако автор учебника не счел целесообразным выбрать в качестве базовой одну из существующих методологий, а продолжает использовать для этих целей собственную методологию. Это объясняется несколькими причинами. Назовем некоторые из них.

1. Несмотря на существенный прогресс, наблюдаемый в последнее время, каждая из нотаций все еще, по мнению автора, обладает серьезными недостатками. Основным из этих недостатков является то, что практически все методологии, реализованные в современных CASE-системах, ориентированы только на реляционные модели и практически не помогают решать задачу проектирования логической структуры базы данных, а позволяют только отображать ее на более абстрактном уровне представления, чем представление в среде конкретной СУБД.

2. Ни одна методология и ни одно CASE-средство не являются явным лидером, чтобы отдать именно им пальму первенства.

3. Алгоритм перехода от ER-модели к структуре базы данных в технической документации и литературе по CASE-системам практически никогда не излагается.

Кроме базовой ER-модели в учебнике подробно рассмотрены возможности построения ER-моделей в среде Design/IDEF и ERWin.

Глава 3 посвящена проектированию логической структуры БД - даталогическому проектированию. Вопросы даталогического моделирования рассмотрены с точностью до класса моделей, а не конкретных СУБД. В этой главе прослеживаются общность всех рассматриваемых моделей и их различия. Используется общий подход к проектированию логической структуры базы данных в среде СУБД, поддерживающих структурированные модели данных разных классов. Понимание методологии проектирования позволяет легко адаптироваться к особенностям конкретных СУБД.

Приведен алгоритм перехода от базовой ER-модели к структуре реляционной базы данных, а также даны дополнительные рекомендации по проектированию реляционных баз данных.

Проектирование в среде СУБД, поддерживающих сетевые модели с разнотипными файлами, иерархические модели вообще в данном учебнике не рассматривались в связи с тем, что они в настоящее время не имеют широкого распространения. При необходимости с основами проектирования БД указанных типов можно ознакомиться в [18, 16 и др.].

Завершается глава характеристикой средств проектирования структуры БД в современных CASE-системах.

Вопросы физического проектирования в среде конкретных СУБД в данном учебнике не затрагивались вообще. Это объясняется тем, что нельзя рекомендовать для обучения какую-то конкретную СУБД (каждая из них имеет специфику в физической организации данных), а также тем, что вопросы физического проектирования для современных реляционных СУБД не столь разнообразны и актуальны, как для СУБД, поддерживающих сетевые и иерархические модели, и во многих системах вообще скрыты от пользователя.

В главе 4 обсуждены теоретические вопросы обеспечения целостности, баз данных: дана классификация ограничений целостности, показана их связь с ER-моделированием. Более конкретно вопросы реализации ограничений целостности рассмотрены в главах 5-7 и 10.

В главе 5 освещены некоторые проблемы создания и ведения баз данных. В нее включены вопросы, связанные с описанием БД, заданием ограничений целостности и обеспечением их соблюдения при обработке баз данных, организацией ввода информации в базы данных. Вопросы создания БД рассмотрены на примере СУБД Access 2000.

Главы 6 и 7 посвящены современным языкам запросов. В главе 6 описаны общие принципы задания запросов в табличном языке QBE, подробно рассмотрена реализация этого языка в системе Access, а также показаны отличия в форме представления запросов на QBE для некоторых других систем.

Глава 7 знакомит с языком SQL. Приведены общая характеристика языка и краткая информация о стандартах SQL. Описаны основные возможности языка в соответствии со стандартом SQL-92. Обращено внимание на вопросы задания ограничений целостности при использовании SQL. Рассмотрены особенности реализации SQL в СУБД Access.

В главе 8 отражены вопросы создания экранных форм ввода-вывода информации и страниц доступа с использованием соответствующих генераторов в среде СУБД Access 2000.

В главе 9 обсуждены вопросы создания отчетов с использованием современных СУБД на примере СУБД Access 2000.

В главе 10 рассмотрены особенности работы с базами данных в многопользовательском режиме, современные технологии организации распределенных банков данных, в том числе и технологии тиражирования, показаны сетевые возможности некоторых СУБД, а также особенности создания и экс-

плуатации БнД, функционирующих в локальных сетях. Достаточно большое внимание уделено вопросам обеспечения целостности и безопасности данных в распределенных банках данных.

Вопросы проектирования и использования баз данных включены в учебные планы практически для всех специальностей. В данном учебнике материалы изложены в соответствии с программой курса «Базы данных» для специальности 0603 «Информационные технологии в экономике». Учебные планы для разных и даже для одноименных специальностей в разных учебных заведениях могут иметь отличия. Это накладывает отпечаток на содержание и структуру учебника. Так, традиционно особенности логических моделей данных, способы физической организации данных, а также теория нормализации рассматриваются в дисциплине «Проектирование баз данных». В МЭСИ, где автор работает профессором, эти вопросы вынесены в курс «Теория экономических информационных систем». В связи с этим указанные вопросы не рассматриваются в данном учебнике.

Курс «Базы данных» тесно связан с другими дисциплинами, изучающими основы вычислительной техники, программирование, теорию множеств и теорию графов, проектирование информационных систем. Особенно тесная связь наблюдается с дисциплиной «Теория экономических информационных систем». Данный учебник написан в предположении, что студенты владеют соответствующим понятийным аппаратом, а также знакомы с теорией нормализации реляционных отношений.

Учебник может быть использован и для студентов других специальностей, связанных с проектированием информационных систем и программного обеспечения, а также для обучения работе с базами данных в качестве конечного пользователя. Причем в зависимости от специальности, для которой ведется преподавание, и структуры учебных планов круг затрагиваемых вопросов и глубина их освещения будут меняться. Для всех специальностей необходимо изучить главу 1. Также для всех специальностей представляют интерес вопросы моделирования предметных областей, поскольку именно они обеспечивают взаимопонимание всех категорий специалистов. Для студентов, не специализирующихся в области информационных технологий, нет необходимости глубокого сравнения методик ER-моделирования, а знакомство со сводной характеристикой CASE-средств и критериями их выбора желательно.

Так как автоматизированные средства проектирования БД распространены не повсеместно, а также вследствие того, что при проектировании небольших простых систем часто проще создать БД вручную, целесообразно изучение какой-либо методики проектирования структуры БД. Изучать основы проектирования БД следует даже в случае, если будут использоваться автоматизированные средства проектирования. Это позволит оценить решение, полученное автоматизированным путем, и при необходимости исправить его; понимание сущности принимаемых проектных решений важно для выбора конкретного решения из некоторого множества альтернатив (многие CASE-средства предоставляют такие возможности). Кроме того, понимая суть мо-

делирования предметной области и алгоритмов перехода от ER-модели к структуре БД, можно, меняя исходную концептуальную модель, воздействовать на получаемое проектное решение.

Язык QBE является языком конечного пользователя, но это не означает, что специалисты в области информационных технологий не должны с ним знакомиться (тем более что это не займет много времени). Основными доводами в пользу этого являются:

- изучение языков запросов разных классов позволяет понять их общность;
- QBE часто используется как «построитель» запросов на SQL;
- часто QBE является самым простым и быстрым способом формирования запроса;
- специалисту по информационным технологиям просто непозволительно не знать то, что знает конечный пользователь.

Для всех специалистов в области информационных технологий необходимо изучение SQL, который является стандартом языков запросов для реляционных СУБД. Для студентов иных специальностей эти вопросы могут быть опущены либо рассмотрены только на уровне общих концепций языка.

Как указывалось выше, данный учебник может быть использован для студентов разных специальностей, поэтому трудно выбрать предметную область, которая была бы одинаково интересна и понятна всем. Приемы же, используемые при построении баз данных, универсальны и не сильно зависят от предметной области. Чтобы иллюстрирующие примеры были понятны студентам разных специальностей, в качестве предметной области взят абстрактный вуз.

Настоящее издание имеет существенные отличия от ранее вышедших книг (С.М. Диго. Проектирование баз данных, 1988, и Проектирование и использование баз данных, 1995). В нем, в частности, подробно рассмотрены вопросы использования CASE-систем Design/ IDEF и ERWin для проектирования баз данных; предложен алгоритм перехода от базовой ER-модели к ER-моделям, используемым в этих системах; вопросы создания БД, запросов, экранных форм и отчетов проиллюстрированы на примере СУБД Access (в предыдущем издании это была СУБД dBase IV); изложен стандартный SQL-92 и др.

Глава 1 ВВЕДЕНИЕ В БАНКИ ДАННЫХ

1.1. Понятие банка данных

Банк данных (БнД) является современной формой организации хранения и доступа к информации. *«Банк данных* – это система специальным образом организованных данных *(баз данных),* программных, технических, языковых, организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных»*.

В этом определении, с одной стороны, подчеркивается, что банк данных является сложной системой, включающей в себя все обеспечивающие подсистемы, необходимые для функционирования любой системы автоматизированной обработки данных. С другой стороны, здесь также обозначены и основные *отличительные особенности* банков данных от иных типов информационных систем, например, реализующих позадачный подход к их созданию.

1. Базы данных создаются обычно для многоцелевого использования, а не для решения какой-либо одной задачи.

2. Базы данных отражают определенную часть реального мира. Надо стремиться, чтобы вся информация, описывающая предметную область, фиксировалась в базе данных однократно, накапливалась и поддерживалась в актуальном состоянии централизованно, а все пользователи, которым эта информация нужна, должны иметь возможность работать с ней.

3. Базы данных – это специальным образом организованные данные. Эти особенности в организации данных заключаются, прежде всего, в том, что БД представляют собой системы взаимосвязанных данных, единство и целостность которых поддерживается специальными программными средствами.

4. Для функционирования БнД необходимо наличие специальных языковых и программных средств (называемых СУБД – система управления базами данных), облегчающих для пользователей выполнение всех операций, связанных с организацией хранения данных, их корректировки и доступа к ним.

Нельзя сказать, что в рассматриваемой нами сфере установилось терминологическое единство. Так, в англоязычной литературе термин «банк данных» используется значительно реже, чем «система баз данных» (database system), который по своему содержанию близок к приведенному выше определению банка данных (система баз данных включает базу данных, систему управления базами данных, соответствующее оборудование и персонал) [36]. Согласно семантике русского языка «система баз данных» воспринимается уже, чем то, что это понятие обозначает в действительности. Поэтому слово

* Общеотраслевые руководящие материалы по созданию банков данных. -М.: ГКНТ, 1982.

«банк» является в этом смысле лучше, так как «банк» привычно обозначает не только то, что хранится в нем, но и всю инфраструктуру (например, «банк» как финансовое учреждение – это ведь не просто «куча денег»). Очевидно, что нельзя отождествлять понятия «база данных» и «банк данных», как это иногда делается в некоторых литературных источниках.

Терминологические различия наблюдаются и при определении других понятий в области БнД. Особенно это касается терминов, используемых в конкретных программных системах. В связи с тем, что терминология конкретных СУБД сильно различается, нельзя описать общие принципы построения БнД, пользуясь терминологией какой-либо одной из них. В учебнике будут введены термины, которые, по мнению автора, в наибольшей степени соответствуют отображаемым ими понятиям.

Следует отметить, что использование тех или иных терминов зависит от аспекта рассмотрения изучаемой проблемы. Так, например, в законе* под базой данных понимается практически любая совокупность данных, которая может быть обработана с помощью ЭВМ. И это оправданно, так как права собственности и иные права не могут зависеть от того, с помощью какого программного средства созданы файлы и какой у них способ организации. Но такое широкое толкование термина БД в курсе «Проектирование баз данных» приведет к нивелированию особенностей банков данных как особой информационной технологии.

Преимущества БнД. Особенности «банковской» организации данных обусловливают их основные преимущества перед «небанковской» организацией.

Наличие единого отображения определенной части реального мира позволяет обеспечить непротиворечивость и целостность информации, возможность обращаться к ней не только при решении заранее предопределенных задач, но и с нерегламентированными запросами. Интегрированное хранение сокращает избыточность хранимых данных, что приводит к сокращению затрат не только на создание и хранение данных, но и на поддержание их в актуальном состоянии.

Использование БнД при правильной его организации должно существенно изменить деятельность организации, где он внедряется, привести к обеспечению большей доступности данных для всех категорий сотрудников, сокращению документооборота, возможности получения разнообразных по форме и содержанию документов, перераспределению функций между сотрудниками и изменению характера выполняемых ими функций и, как следствие, улучшить всю систему управления предприятием.

Централизованное управление данными также дает целый ряд преимуществ. Использование СУБД обеспечивает высокое качество выполнения

---

* Закон «О правовой охране программ для электронных вычислительных машин и баз данных» от 23.09.92 № 3523-1.

функций по управлению данными и облегчает процесс создания информационных систем (ИС).

Выделение специальной группы сотрудников, выполняющих функции по проектированию и развитию БнД (администраторов БД), и освобождение от этих функций всех остальных пользователей не только приводит к снижению требований к остальным участникам процесса создания и функционирования БнД, но и повышает качество разработок, так как вопросами организации данных занимается небольшое число профессионалов в этой области.

Преимуществом банков данных является также то, что они обеспечивают возможность более полной реализации принципа независимости прикладных программ от данных, чем это возможно при организации локальных файлов.

Пользователи БнД. В процессе создания и эксплуатации БнД с ним взаимодействуют пользователи разных категорий (рис. 1.1). Базы данных создаются для удовлетворения потребностей конечных пользователей. Чаще всего это специалисты конкретных предметных областей, использующие БД для выполнения своих профессиональных обязанностей. В последнее время БД все чаще применяются и для удовлетворения непроизводственных информационных потребностей. Конечные пользователи – наиболее многочисленная группа пользователей. Нельзя недооценивать важность этой группы и не понимать ее характерных особенностей.

Специфическими пользователями БнД являются сотрудники информационных служб. Они работают в основном с метаинформацией. Часто бывает желательно, чтобы другая информация была для них закрыта. Кроме того, они используют и другие ресурсы БнД для выполнения своих функций.

Категория «Конечные пользователи» неоднородна. Конечные пользователи различаются широтой информационных потребностей, квалификацией, режимами взаимодействия с БнД и др. Это могут быть случайные пользователи, обращающиеся к базе данных время от времени, а могут быть и регулярные пользователи. Конечные пользователи могут отличаться друг от друга и степенью владения вычислительной техникой. От конечных пользователей не должно требоваться каких-то специальных знаний в области вычислительной техники и языковых средств.

При создании БнД важно не только построение классификационной схемы, но и распределение реальных конечных пользователей по группам, так как от характеристики пользователей будут зависеть принимаемые проектные решения.

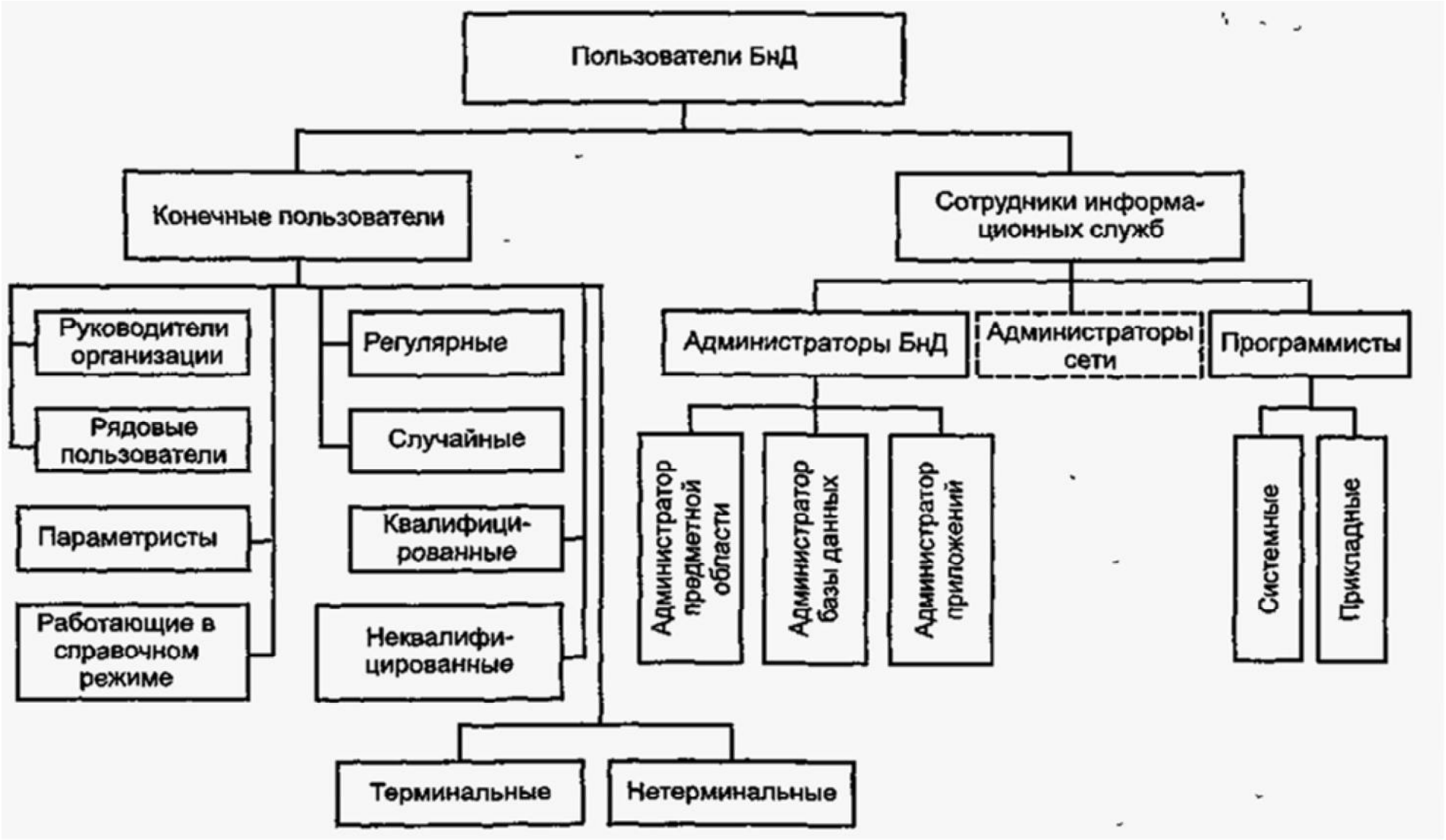


Рис.1.1. Классификация пользователей БнД

Поскольку использование БнД оказывает влияние на все аспекты деятельности организации, особую роль играют *руководители организации.* Именно они должны обеспечить проведение единой информационной политики и организацию взаимодействия различных подразделений через общую базу данных. Они должны создавать подразделения, отвечающие за создание и функционирование БнД, определять функциональные обязанности сотрудников, которые существенно изменятся с внедрением БнД. Кроме того, руководители организации выступают в качестве конечных пользователей с наиболее высоким приоритетом.

Отдельные пользователи в процессе работы с базой данных могут менять содержание БД – это так называемые *параметристы.* Другие могут только использовать хранящуюся в БД информацию.

Пользователи могут взаимодействовать с БД как непосредственно *(терминальные* пользователи), так и через посредников *{нетерминальные).*

Понятием «конечные пользователи» определяются не только отдельное лицо или группа лиц, но и вычислительные процессы/задачи, а иногда и целые системы, взаимодействующие с БнД.

В зависимости от особенностей создаваемого банка данных круг его конечных пользователей может существенно различаться.

Категория «сотрудники информационных служб» также является неоднородной. В рамках курса «Базы данных» наибольший интерес для нас представляют *«Администраторы БнД»* - лица, ответственные за создание БнД и его надежное функционирование, за соблюдение регламента доступа к хранимым данным, за развитие БнД.

Наличие в составе СУБД средств, ориентированных на разные категории пользователей, делает возможной работу с базой данных не только специалистов в области обработки данных, но практически любого пользователя, причем это использование может быть как для их профессиональных целей, так и для удовлетворения потребности в информации в быту и т.п.

Предпосылки широкого использования БнД. Очевидные преимущества БнД и объективные предпосылки их создания привели к широкому их использованию. К числу предпосылок применения БнД относятся следующие:

- объекты реального мира находятся в сложной взаимосвязи между собой. Это приводит к необходимости, чтобы их информационное отражение также представляло единое взаимоувязанное целое;
- информационные потребности различных пользователей существенно пересекаются, что делает целесообразным использование единых баз данных и обеспечение доступа к ним разных пользователей (рис. 1.2);
- функции создания и ведения информационного фонда и предоставления нужных данных тем или иным процессам являются универсальными, общими при решении разнообразных задач. Создание специализированных программных средств для управления данными приво-

дит к повышению уровня выполнения этих функций и сокращению трудоемкости создания информационных систем;

- современный уровень развития технического и программного обеспечения, а также теории и практики построения информационных систем позволяет создавать эффективные БнД.

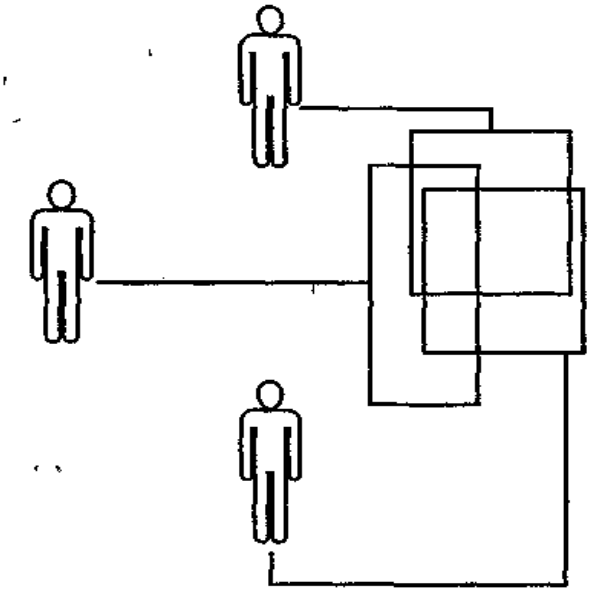


Рис. 1.2. Пересечение информационных потребностей пользователей

Требования к БнД. Особенности «банковской» организации данных позволяют сформулировать основные требования, предъявляемые к БнД:

- адекватность отображения предметной области (полнота, целостность и непротиворечивость данных, актуальность информации, т.е. ее соответствие состоянию отображаемой реальной системы на данный момент времени);
- возможность взаимодействия пользователей разных категорий и в разных режимах, обеспечение высокой эффективности доступа для разных приложений;
- дружелюбность интерфейсов и малое время на освоение системы, особенно для конечных пользователей;
- обеспечение секретности и конфиденциальности для некоторой части данных, определение групп пользователей и их полномочий;
- обеспечение взаимной независимости программ и данных;
- обеспечение надежности функционирования БнД; защита данных от случайного и преднамеренного разрушения; возможность быстрого и полного восстановления данных в случае их разрушения; технологичность обработки данных;
- приемлемые характеристики функционирования БнД (стоимость обработки, время реакции системы на запросы, требуемые машинные ресурсы и др.).

Недостатки БнД. Недостатки БнД вытекают из их достоинств. Создание интегрированной системы, естественно, сложнее, чем создание множества локальных систем. Как следствие, предъявляются высокие требования к квалификации разработчиков БнД. В результате интеграции возможна некоторая потеря эффективности отдельных приложений (но общая эффективность всей системы будет выше). Для управления данными требуется специализированное программное обеспечение, которое в зависимости от класса системы может быть сравнительно дорогим, предъявляющим повышенные требования к техническим средствам. Эксплуатация распределенных корпоративных БнД – процесс сложный и дорогостоящий.

Но, несмотря на некоторые недостатки, присущие такой форме организации данных, преимущества БнД значительно превосходят их. Кроме того, имеется очень широкий круг СУБД разных классов и технологий их использования. Правильный выбор системы позволит свести отрицательные последствия к минимуму.

1.2. Компоненты банка данных

Банк данных является сложной человеко-машинной системой, включающей различные взаимосвязанные и взаимозависимые компоненты (рис. 1.3).

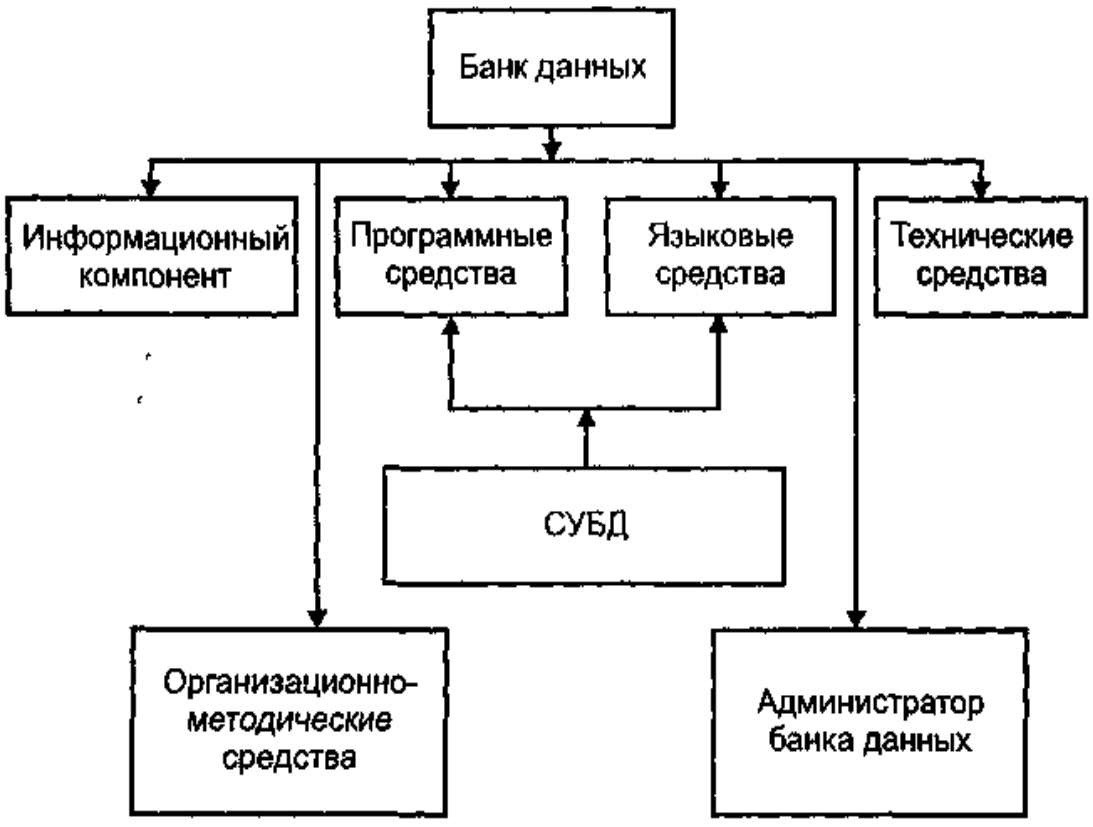


Рис. 1.3. Компоненты БнД

1.2.1. Информационный компонент

Ядром БнД является база данных. *База данных* - это поименованная совокупность взаимосвязанных данных, находящихся под управлением СУБД.

Существует множество определений базы данных. Некоторые из них имеют право на существование. Другие устарели и не соответствуют современным представлениям о БД. Так, в ранних определениях баз данных указывалось на отсутствие дублирования данных в них. На самом деле это не так. В базах данных может наблюдаться дублирование информации. Оно может быть вызвано спецификой используемой модели данных, не позволяющей полностью устранить дублирование, или технологическими причинами (обеспечение большей надежности, сокращение времени реакции системы и др.). Но это должно быть «управляемое дублирование», причины и цели возникновения которого известны администратору базы данных и отслеживаются как им, так и СУБД.

В настоящее время действует Закон РФ «О правовой охране программ для электронных вычислительных машин и баз данных». В этом Законе в ст. 1 дается следующее определение базы данных: «База данных - это объективная форма представления и организации совокупности данных (например, статей, расчетов), систематизированных таким образом, чтобы эти данные могли быть найдены и обработаны с помощью ЭВМ». Если учесть назначение этого закона, вполне естественно, что в нем сделан иной акцент, чем в данном нами определении БД; определение, используемое в тексте данного Закона, является более широким, чем приведенное нами.

Мы в качестве рабочего будем пользоваться приведенным выше определением и не всякие файлы будем считать базами данных.

В технической документации некоторых СУБД, а также в некоторых литературных источниках в состав БД включаются не только собственно хранимые данные о предметной области, но и описания БД. Более правильно описания баз данных считать самостоятельными компонентами БнД, даже если они и хранятся вместе с самими данными.

Описания баз данных относятся к *метаинформации,* т.е. информации об информации. Описание баз данных часто называют *схемой.* Кроме того, в БнД могут присутствовать описания отдельных частей базы данных с точки зрения конкретных пользователей. Такое описание называется *подсхемой.*

Хотя в буквальном смысле *метаинформация* — это информация об информации, в информационных системах это понятие трактуется более широко. Кроме описания баз данных в состав метаинформации, хранимой в БнД, может включаться информация о предметной области, необходимая для проектирования автоматизированной информационной системы, о пользователях БнД, о проектных решениях и некоторая другая информация (рис. 1.4).

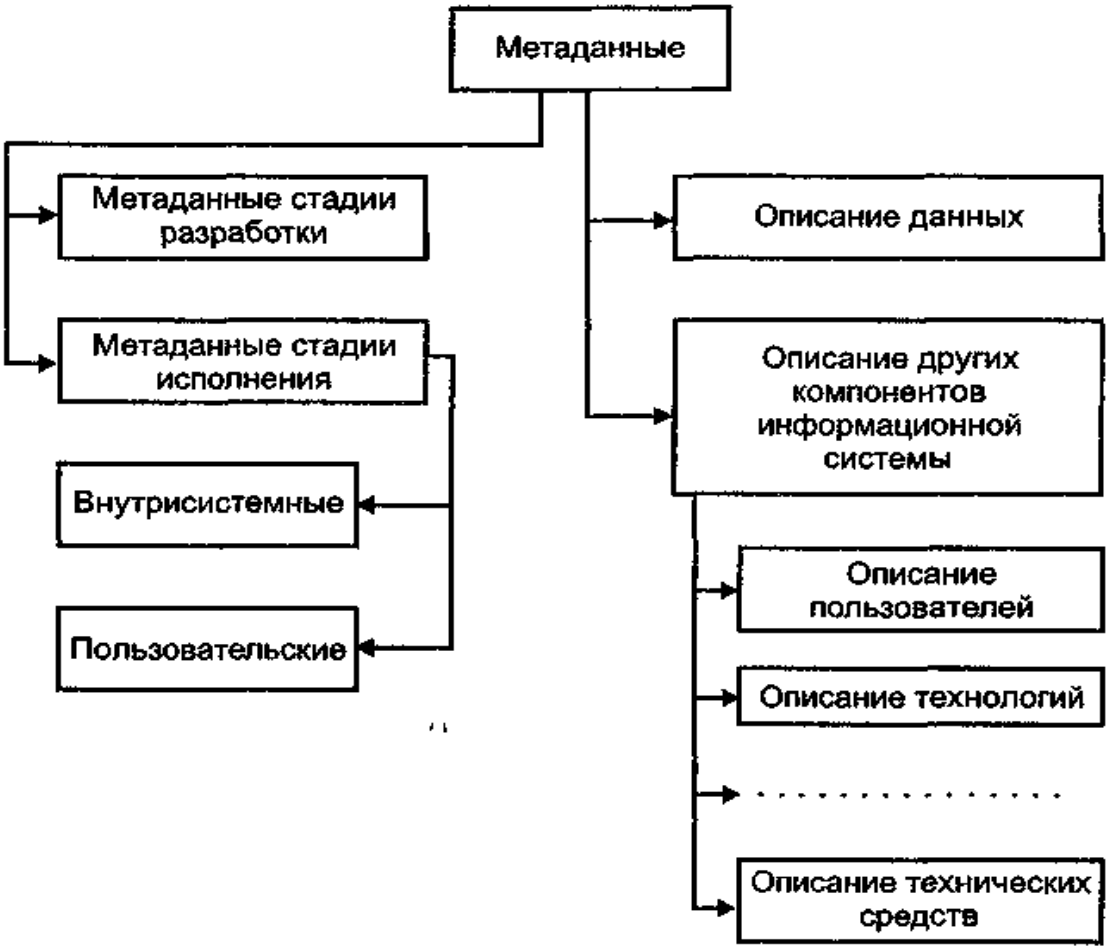


Рис. 1.4. Классификация метаданных

Централизованное хранилище метаинформации называется *словарем данных.* В литературе используются также термины словарь-справочник, энциклопедия, репозиторий. В некоторых источниках выявляются различия между этими терминами, в других - они используются как синонимы. Нам для данного уровня рассмотрения эти различия несущественны.

Роль словарной системы особенно возрастает при использовании средств автоматизированного проектирования информационных систем. Для большинства из них репозиторий является ядром всей системы. Кроме того, роль репозитория особо значима в распределенных системах.

К банку данных не относятся немашинные документы, служащие источниками информации, вводимой в БД, файлы входной и выходной информации, архивные файлы, выходные документы. Однако многие СУБД включают в свой состав языковые средства для описания этих компонентов. В этом случае сами описания, используемые в процессе функционирования БнД, будут входить в его состав.

Как уже отмечалось выше, терминология, используемая в разных системах и разных литературных источниках, существенно различается. Так, в ранних версиях многих «настольных» реляционных СУБД вообще не использовался «механизм» базы данных (т.е. создавались фактически отдель-

ные реляционные таблицы, каждая из которых запоминалась в отдельном файле базы данных). Во многих литературных источниках каждый такой файл стали называть базой данных, что неправильно.

В некоторых системах, например Access, под БД понимают совокупность разных объектов: таблиц, запросов, форм, отчетов, макросов и модулей, т.е. понятие базы данных расширено и включает в себя практически все информационные компоненты, созданные для конкретного приложения. В других системах, в частности в Paradox, для обозначения подобной совокупности взаимосвязанных объектов используется понятие «семейство», что, очевидно, терминологически более правильно.

При работе с конкретной системой надо, прежде всего, уточнить терминологию, используемую в ней.

1.2.2. Программные средства БнД

Программные средства БнД представляют собой сложный комплекс, обеспечивающий взаимодействие всех частей информационной системы при ее функционировании (рис 1.5).

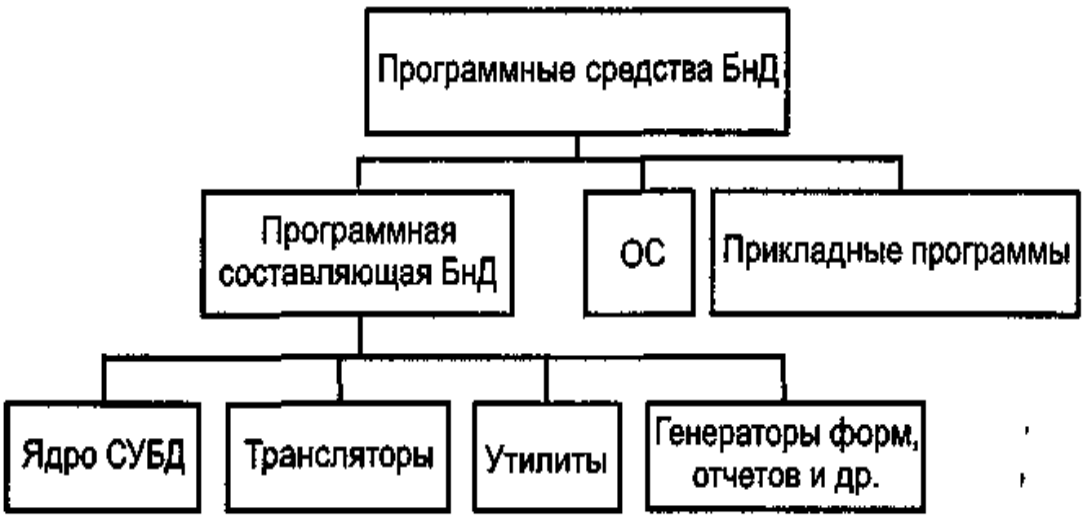


Рис. 1.5. Программные средства БнД

Основу программного обеспечения БнД составляют программные компоненты СУБД. Среди них можно выделить ядро СУБД, обеспечивающее создание БД, организацию ввода, обработки и хранения данных, т.е. именно то, что называется «управлением данными», а также другие компоненты, обеспечивающие настройку системы, средства тестирования, утилиты, обеспечивающие выполнение вспомогательных функций, таких, как восстановление баз данных, сбор статистики о функционировании БнД и др. Важным компонентом СУБД являются трансляторы или компиляторы для используемых ею языковых средств.

В состав большинства СУБД включены программные компоненты, позволяющие автоматизировать проектирование систем обработки информации (генераторы отчетов, меню и др.). Строго говоря, эти функции не являются непосредственно функциями по управлению данными, но большинство современных программных средств, которые продолжают называться СУБД, выходят за названные рамки и фактически являются мощными комплексными инструментальными средствами, позволяющими автоматизировать процесс создания информационных систем.

Подавляющее большинство СУБД работает в среде универсальных операционных систем (ОС) и взаимодействует с ОС при обработке обращений к БнД. Поэтому можно считать, что ОС также входит в состав БнД.

Для удовлетворения конкретных потребностей пользователей пишутся соответствующие программы*, которые представляют прикладное программное обеспечение БнД.

В зависимости от используемых технологий создания и функционирования систем могут появляться те или иные дополнительные компоненты. Так, при использовании CASE-технологий будут присутствовать соответствующие программные компоненты, поддерживающие проектирование и перепроектирование системы.

При работе в архитектуре «клиент-сервер» программные средства будут подразделяться на соответствующие компоненты: клиентская часть, обеспечивающая интерфейс пользователя с системой, серверная часть, реализующая обработку запроса на сервере, и связная часть, обеспечивающая взаимодействие элементов в сети.

Программные средства, используемые при создании и эксплуатации БнД, будут также зависеть от масштаба БнД и требований, предъявляемых к нему.

1.2.3. Языковые средства БнД

Языковые средства СУБД являются важнейшим компонентом банков данных, так как, в конечном счете, они обеспечивают интерфейс пользователей разных категорий с банком данных. Набор используемых языковых средств широк и разнообразен. Языковые средства, используемые в БнД, можно классифицировать по разным признакам (рис. 1.6).

Языковые средства большинства современных СУБД относятся к языкам 4-го поколения (к 1-му поколению языков относят машинные языки, ко 2-му - символические языки ассемблера, к 3-му - алгоритмические языки типа PL, Cobol и т.п., которые в 1960-е гг. назывались языками высокого уровня, но уровень, которых гораздо ниже, чем у языков 4-го поколения. Имеют-

* Понятие «программа» здесь трактуется широко. Это могут быть, например, и экранные формы, созданные с использованием визуальных средств, и запросы, написанные на любом языке запросов.

ся еще и языки 5-го поколения, к которым относят языки систем искусственного интеллекта, например Prolog).



Рис. 1.6. Классификация языковых средств БнД

Языки 4-го поколения создавались по принципу: «люди стоят дороже, чем машины» [44]. При их проектировании использовались следующие принципы.

1. Принцип минимума работы: язык должен обеспечить минимум усилий, чтобы «заставить» машину работать.

2. Принцип минимума мастерства: работа должна быть так проста, как только это возможно; она не должна быть уделом избранных и быть понятной лишь посвященным.

3. Принцип естественности языка, упразднения «инородного» синтаксиса и мнемоники. Язык не должен требовать от пользователей значительных усилий в изучении синтаксиса или содержать много мнемонических или иных обозначений, которые быстро забываются.

4. Принцип минимума времени. Язык должен позволять без существенной задержки реализовывать возникающие потребности в доступе к информации и ее обработке.

5. Принцип минимума ошибок. Технология должна быть спроектирована таким образом, чтобы минимизировать ошибки человека, а уж если они возникли, то, по возможности, «выловить» их автоматически.

6. Принцип минимума поддержки. Механизм языка должен позволять легко вносить изменения в имеющиеся приложения.

7. Принцип максимума результата. Язык предоставляет пользователям мощный инструмент для решения разнообразных задач.

На рис. 1.7 приведены компоненты языка 4-го поколения. Как видим, здесь представлены все основные «генераторы», наличие которых уже стало традиционным для СУБД разных классов.

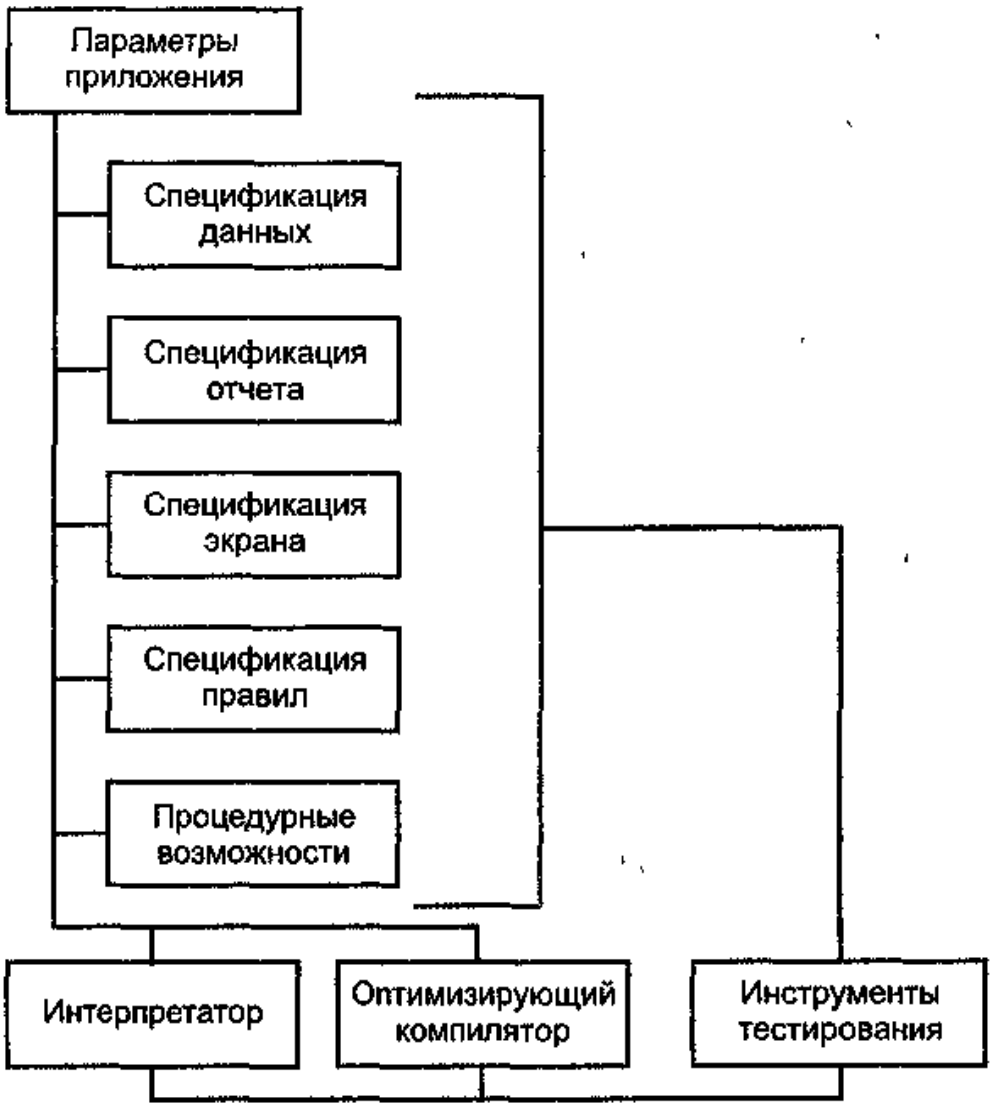


Рис. 1.7. Компоненты языка 4-го поколения

Можно выделить две концепции развития языковых средств: концепцию разделения и концепцию интеграции. При использовании концепции *разделения* различают языки описания данных (ЯОД) и языки манипулирования данными (ЯМД). Назначение каждого из этих подклассов ясно из их названия.

Иногда в особую группу выделяют языки запросов (ЯЗ). Первоначально под языками запросов понимали языки высокого уровня, ориентированные на конечного пользователя, предназначенные для формирования запросов к БД (в такой трактовке их можно считать одной из разновидностей ЯМД). Однако сейчас ЯЗ понимается шире *(см.* главу 3); многие ЯЗ включают в себя еще и возможности описания данных и корректировки БД.

В составе *языков описания данных* в зависимости от особенностей СУБД поддерживаются все или некоторые из следующих языков: язык описания схем (ЯОС), язык описания подсхем (ЯОПС), язык описания хранимых данных (ЯОХД), языки описания внешних данных (входных, выходных). В некоторых СУБД и сами эти разновидности языков, и создаваемые с их помощью элементы ИС являются самостоятельными компонентами, в других - некоторые из них могут быть объединены.

Языки *манипулирования данными* разделяются на две большие группы: процедурные и непроцедурные (декларативные). При пользовании процедурными языками надо указать, какие действия и над какими объектами необходимо выполнить, чтобы получить результат. В непроцедурных языках указывается, что надо получить в ответе, а не как этого достичь.

Процедурные языки могут различаться по основным информационным единицам, которыми они манипулируют. Это могут быть языки, ориентированные на позаписную обработку данных, и языки, ориентированные на операции над множеством записей. Так, операции реляционной алгебры оперируют целиком отношением, а не каждой его записью. Примерами непроцедурных языков являются языки, основанные на реляционном исчислении. К таким языкам относится широко используемый язык запросов SQL. Табличный язык QBE также является непроцедурным языком.

Языковые средства предназначаются для пользователей разных категорий: конечных пользователей, системных аналитиков, профессиональных программистов. Повышение уровня языковых средств, их дружелюбности приводит к тому, что все большее число функций выполняется пользователями-непрограммистами самостоятельно, без посредников.

По *функциональным возможностям* выделяют следующие категории языков [44].

1. Языки, имеющие только возможности запросов. Они обеспечивают вывод требуемых данных на экран или печать в нужном формате. В настоящее время используются редко.

2. Комплексные языки запросов/обновлений. Это более развитые языки; они позволяют формулировать сложные запросы, относящиеся к нескольким взаимосвязанным записям, и обновлять данные так же легко, как и формули-

ровать запросы. Используя их, пользователи могут создавать свои собственные файлы.

3. Генераторы отчетов. Они дают возможность выбирать нужные данные из файлов или баз данных и форматировать их в виде требуемых форм документов.

4. Графические языки. Использование графических средств в настоящее время постоянно расширяется. С их помощью можно выводить данные в виде различных графиков и диаграмм, а также использовать другие изобразительные возможности. Как и генераторы отчетов, графические языки позволяют осуществлять отбор информации из файлов или баз данных по различным критериям, а также выполнять арифметические и логические манипуляции с данными.

5. Инструментальные средства поддержки решений. Языки этого типа предназначены для создания систем принятия решений. Это могут быть системы типа «что-если», системы, выполняющие временной или трендовый анализ, и др. Возможно использование как универсальных, так и проблемно-ориентированных средств.

6. Генераторы приложений. Обеспечивают возможность описания непроцедурным путем требуемой обработки информации и дальнейшей автоматической генерации программ.

7. Машинно-ориентированные языки спецификаций. Фактически являются генераторами приложений, дальнейшим их развитием. В отличие от генераторов приложений языки спецификаций более универсальны и позволяют специфицировать приложения разных типов.

8. Языки очень высокого уровня. В большинстве случаев приложения строятся с помощью непроцедурных языков. Однако некоторые языки являются процедурными (например, NOMAD), но программирование на них значительно короче, чем, например, на Cobol.

9. Параметризированные пакеты прикладных программ (ППП). Эта категория программных средств известна давно, и «4-е поколение» относится к таким ППП, которые допускают легкую модификацию самого пакета, позволяют пользователям генерировать собственные отчеты, запросы к базе данных и т.д.

10. Языки приложений. Многие языки 4-го поколения являются универсальными, другие - спроектированы для специфических приложений. Примерами таких языков являются языки для управления финансами, управления работой станков с программным управлением и т.д.

По *форме представления* различают аналитические, табличные и графические языковые средства. Классификация языковых средств по этому признаку относится как к языкам описания данных, так и к языкам манипулирования данными. Так, описание таблицы с использованием команды CREATE TABLE языка SQL *(см.* главу 7) является примером аналитической формы ЯОД, а описание такой же таблицы в Access *(см.* главу 5) и большинстве других настольных СУБД - примером табличной формы описания данных. В качестве примеров табличной и аналитической форм задания запро-

сов можно привести языки QBE *(см.* главу 6) и SQL *(см.* главу 7) соответственно.

Достаточно часто бывает, что в рамках одной СУБД для одних и тех же целей могут использоваться языки разных типов. Так, например, во многих СУБД (dBase, FoxPro и др.) для манипулирования данными могут использоваться:

- табличный язык запросов типа QBE;
- язык SQL - аналитический язык запросов, относящийся к классу языков исчисления кортежей;
- процедурный язык программирования (для указанных выше систем dBase, FoxPro — это язык xBase, часть операторов которого реализует операции реляционной алгебры, а другая часть, более значительная по количеству операторов и функций, выполняет нереляционные операции, обеспечивающие позаписную обработку файлов, организацию циклической и условной обработки, ввод-вывод данных, корректировку, возможность работы с переменными памяти и другие возможности).

Описание данных в этих системах может быть представлено в табличном виде либо, если определение данных выполняется средствами SQL или с использованием операторов языка xBase, в аналитическом виде.

Кроме упомянутых языковых средств эти системы включают в себя генераторы экранных форм, отчетов и приложений, а также язык разветвленной иерархической системы «меню», позволяющей пользователю выбрать те действия, которые он желает выполнить.

Часто СУБД обеспечивают автоматическое преобразование «текстов» с одного языка на другой. Так, например, многие СУБД, такие, как Access, FoxPro и др., используют языки запросов табличного типа не только для непосредственной реализации запросов, но и как средство для более простого описания запроса и последующего автоматического преобразования его на язык SQL.

1.2.4. Технические средства БнД

Совокупность типов технических средств, на которых реализуются БнД, не отличается от всех иных автоматизированных ИС (рис. 1.8). Это ЭВМ, периферийные средства для ввода информации в базу данных, средства хранения данных и средства отображения выводимой информации. Если банк данных реализуется в сети, то необходимы соответствующие технические средства (показано пунктиром) для обеспечения ее работы. Но банки данных предъявляют свои требования к используемым техническим средствам.

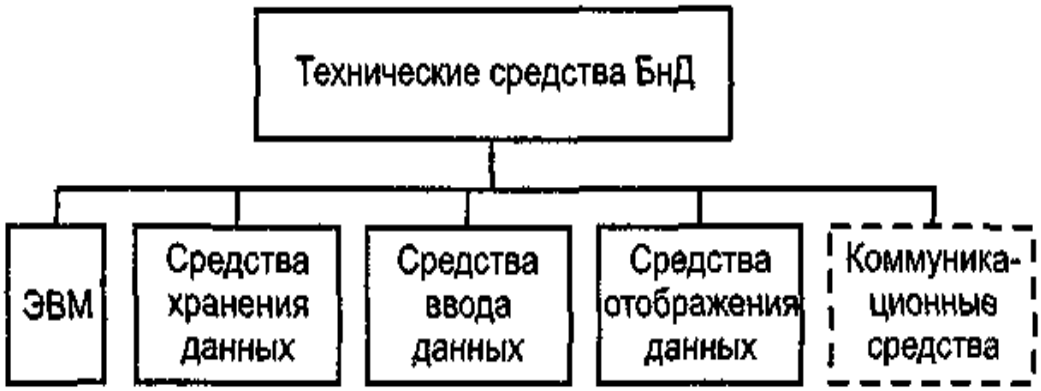


Рис. 1.8. Технические средства БнД

Состав и тип технических средств, на которых реализуются БнД, зависят от многих факторов, основными из которых являются технические характеристики оборудования, используемые технологии обработки данных, масштаб системы, временные ограничения на скорость реакции системы, сложность обработки, стоимостные характеристики и др.

В качестве ЭВМ для банков данных чаще всего используются универсальные компьютеры.

Первоначально БнД реализовывались в основном на больших ЭВМ, а для доступа к БД использовались терминалы. В связи со значительным и постоянным улучшением характеристик персональных ЭВМ появилась возможность реализовать банки данных и на машинах этого класса. Но сначала характеристики персональных ЭВМ были недостаточными, чтобы в полной мере реализовать идеологию банков данных. Стала наблюдаться некоторая раздробленность информационных систем, что, в свою очередь, привело к бурному развитию сетевых технологий и использованию соответствующих технических средств.

Существуют и специализированные технические средства, предназначенные для создания и эксплуатации банков данных (машины баз данных), но они не нашли широкого распространения.

В 1990-е гг. некоторые фирмы (Oracle, Sun) активно развивали идею применения так называемых «сетевых компьютеров». Эти компьютеры представляют собой дешевые рабочие станции без дисковых накопителей*, которые предназначены для работы в сети, и предполагают использование программных средств и данных, находящихся на сервере.

Использование сетевых компьютеров предполагает обязательное применение мощных ЭВМ в качестве серверов, предъявляет высокие требования к организации хранения данных, к качеству каналов связи. При этом во многом становится предопределенной технология обработки данных (особенно в части распределения функций между клиентом и сервером). Применение се-

* Позднее появились предложения создавать сетевые компьютеры с собственными накопителями.

тевых компьютеров обусловлено не столько стремлением сэкономить за счет установки более дешевых компьютеров, сколько желанием упорядочить использование программных средств, упростить систему обработки информации в целом, облегчить и удешевить поддержку системы.

Недостатками такого подхода являются:

- очень большая зависимость от «центральной» системы, потеря самостоятельности конечными пользователями;
- уязвимость системы;
- невозможность/неэффективность обеспечения потребности всех пользователей таким образом (хотя потребности пользователей и пересекаются, но степень пересечения может быть разной; кроме того, не исключается наличие сугубо персональных данных; хранение которых на отдаленном сервере приводит к непроизводительным расходам);
- очень высокие требования к серверной части системы.

Другим новым явлением является использование карманных ПК в качестве коммуникационных устройств для доступа к корпоративным данным в глобальных сетях.

Характеристики карманных компьютеров существенно улучшаются. Для них создается соответствующее программное обеспечение, позволяющее использовать их для мобильных пользователей, работающих в общей системе (тиражирование и синхронизация данных).

Эти компьютеры являются более легкими, что немаловажно для мобильных пользователей, а также стоят дешевле, чем переносные ПК. Ведущие производители СУБД приспосабливают свои крупномасштабные серверные системы для доступа из карманных ПК.

Тип используемых ЭВМ будет зависеть от масштаба создаваемой системы (рис. 1.9). В настоящее время в подавляющем большинстве случаев БнД реализуются в сетевой среде с применением множества разнотипных ЭВМ, причем их состав постоянно меняется в процессе эксплуатации банка данных.

Особую роль для обеспечения эффективного и надежного функционирования банка данных играют средства хранения информации. Память в БнД обычно организуется в виде многоуровневой системы. Необходимо обращать внимание на выбор запоминающих устройств для организации хранения не только данных, предназначенных для оперативного доступа к ним, но и архивных данных.

В банках данных, так же как и во всех других информационных системах, выполняются операции по вводу, хранению, обработке и выводу информации (рис. 1.10). При выполнении каждой из этих операций могут использоваться различные технологии и, как следствие, различные технические и программные средства для их поддержания.



Рис. 1.9. Взаимосвязь используемых ЭВМ и технологии организации ИС

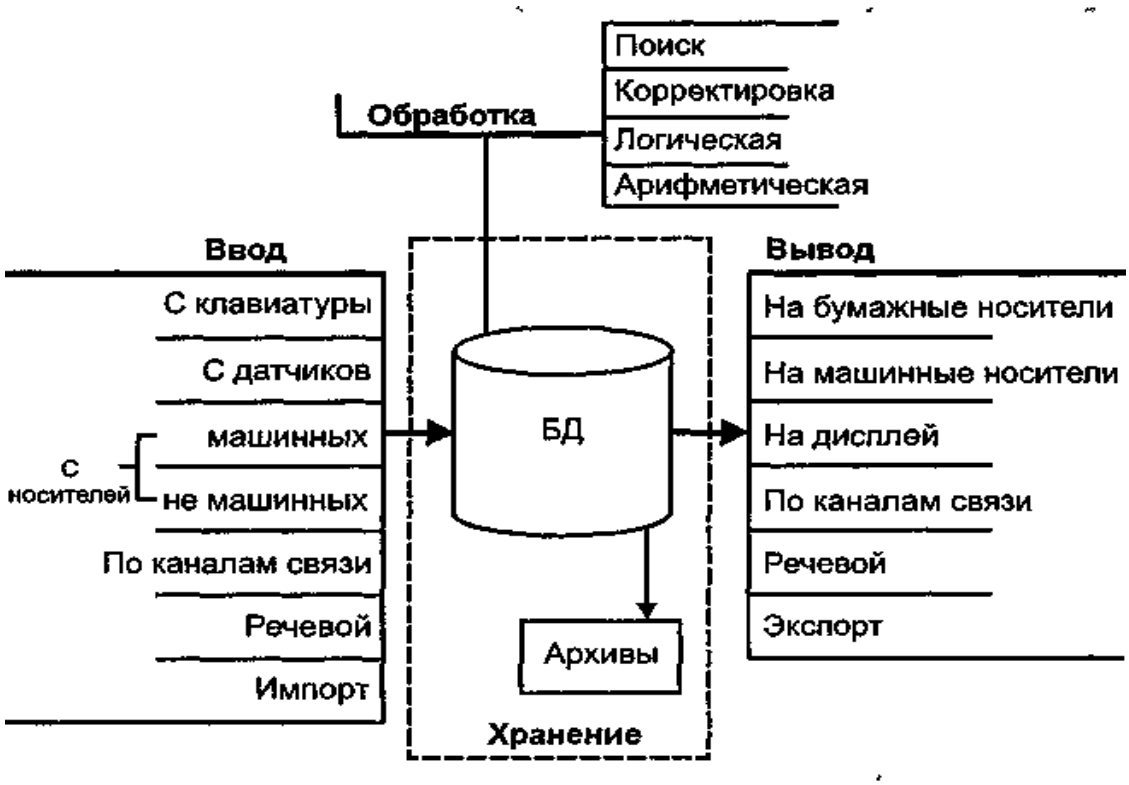


Рис. 1.10. Операции с БД

1.2.5. Организационно-методические средства

Организационно-методические средства банка данных представляют собой различные инструкции, методические и регламентирующие материалы, предназначенные для пользователей разных категорий, взаимодействующих

с банком данных. Это могут быть инструкции конечным пользователям по работе с базой данных, документы, определяющие права доступа и регламент работы; сюда же отнесем и методики проектирования баз.

1.2.6. Администраторы банка данных

Функционирование БнД невозможно без участия специалистов, обеспечивающих создание, функционирование и развитие БнД. Такая группа специалистов называется администратором банка данных (АБнД). Эта группа специалистов считается составной частью банка данных.

В зависимости от сложности и объема банка данных, от особенностей используемой СУБД служба администрации банка данных может различаться как по составу и квалификации специалистов, так и по количеству работающих в этой службе.

Функции администратора банка данных. АБД выполняют работы по созданию и обеспечению функционирования БнД на протяжении всех этапов жизненного цикла системы. В составе группы администраторов банка данных можно выделить различные подгруппы в зависимости от выполняемых ими функций. Численность группы администрации и выполняемые ею функции будут в значительной степени зависеть от масштаба банка данных, специфики хранимой в нем информации, типа банка данных, особенностей используемых программных средств и некоторых других факторов.

В составе администрации БнД должны быть системные аналитики, проектировщики структур данных и внешнего по отношению к банку данных информационного обеспечения, проектировщики технологических процессов обработки данных, системные и прикладные программисты, операторы, специалисты по техническому обслуживанию. Если речь идет о коммерческом банке данных, то важную роль здесь будут играть специалисты по маркетингу.

Администраторы банка данных выполняют большой круг разнообразных функций. Далее будут подробно рассмотрены некоторые из них. Сейчас же просто перечислим основные функции.

1. Анализ предметной области: описание предметной области, выявление ограничений целостности, определение статуса информации, потребностей пользователей, статуса пользователей, соответствия «данные - пользователь», объемно-временных характеристик обработки данных.

2. Проектирование структуры базы данных: определение состава и структуры информационных единиц, составляющих базу данных; задание связей между ними, выбор методов упорядочения данных и методов доступа к информации, описание структуры БД на ЯОД.

3. Задание ограничений целостности при описании структуры базы данных и процедур обработки БД: задание ограничений целостности, присущих предметной области; определение ограничений целостности, вызванных структурой базы данных; разработка процедур обеспечения целостности БД

при вводе и корректировке данных; обеспечение ограничений целостности при параллельной работе пользователей в многопользовательском режиме.

4. Первоначальная загрузка и ведение базы данных: разработка технологии первоначальной загрузки и ведения (изменения, добавления, удаления записей) БД, проектирование форм ввода, создание программных модулей, подготовка исходных данных, ввод и контроль ввода.

5. Защита данных от несанкционированного доступа:

- обеспечение парольного входа в систему: регистрация пользователей, назначение и изменение паролей;
- обеспечение защиты конкретных данных: определение прав доступа групп пользователей и отдельных пользователей, определение допустимых операций над данными для отдельных пользователей, выбор/создание программно-технологических средств защиты данных, шифрование информации в целях защиты данных от несанкционированного использования;
- тестирование средств защиты данных;
- фиксация попыток несанкционированного доступа к информации;
- исследование возникающих случаев нарушения защиты данных и проведение мероприятий по их предотвращению.

6. Защита данных от разрушений. Одним из способов защиты от потери данных является резервирование. Используется как при физической порче файла, так и в случае, если в БД внесены нежелательные необратимые изменения.

7.Обеспечение восстановления БД: разработка программно-технологических средств восстановления БД, организация ведения системных журналов.

8. Анализ обращений пользователей к БД: сбор статистики обращений пользователей к БД, ее хранение и анализ (кто из пользователей, к какой информации, как часто обращался, какие выполнял операции, время выполнения запросов, анализ причин безуспешных (в том числе и аварийных) обращений к БД).

9. Анализ эффективности функционирования БнД и развитие системы: анализ показателей функционирования системы (время обработки, объем памяти, стоимостные показатели), реорганизация и реструктуризация баз данных, изменение состава баз данных, развитие программных и технических средств.

10. Работа с пользователями: сбор информации об изменениях в предметной области, об оценке пользователями работы БнД, определение регламента работы пользователей с БнД, обучение и консультирование пользователей.

11. Подготовка и поддержание системных программных средств: сбор и анализ информации о СУБД и других программных продуктах (ПП), приобретение программных средств, их установка, проверка работоспособности, поддержание системных библиотек, развитие программных средств.

12. Организационно-методическая работа: выбор или создание методики проектирования БД, определение целей и направлений развития системы, планирование этапов развития БнД, разработка и выпуск организационно-методических материалов.

Связи администратора банка данных. В процессе своей деятельности администратор БнД взаимодействует с другими категориями пользователей банка данных, а также и с «внешними» специалистами, не являющимися пользователями БнД (рис. 1.11).

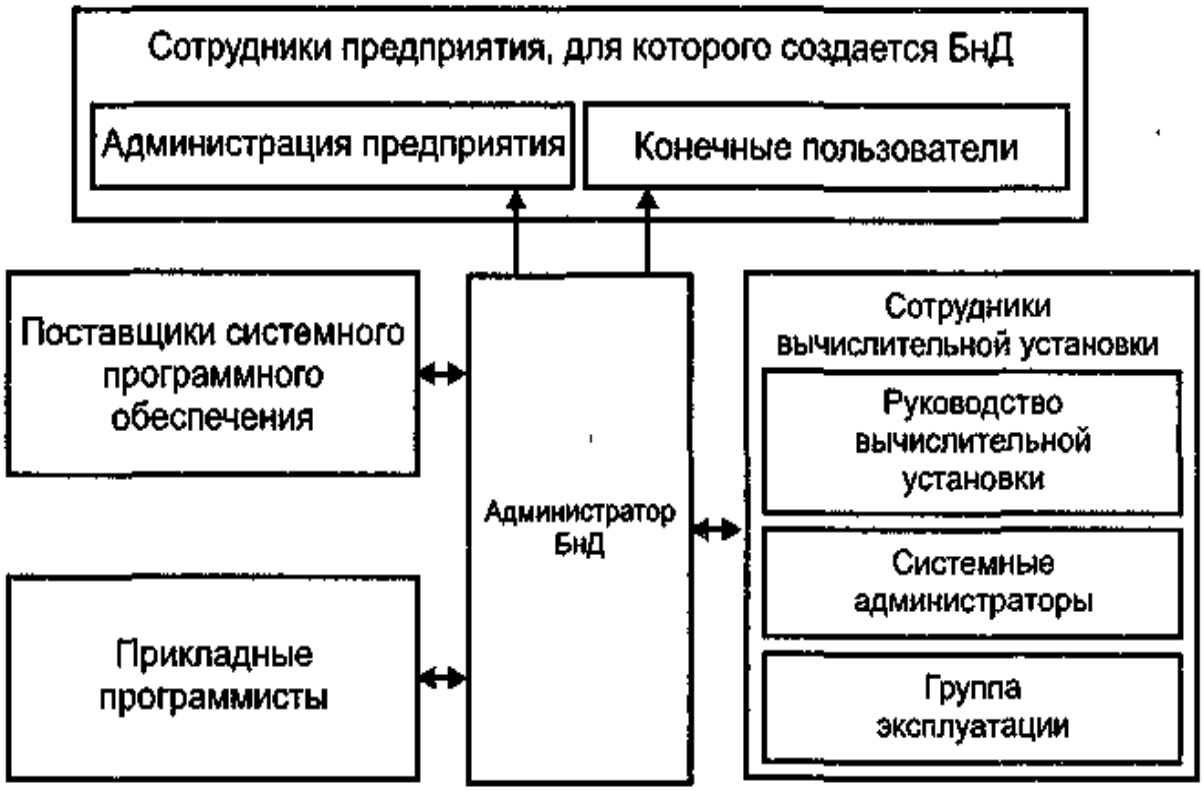


Рис. 1.11. Взаимодействие АБнД с другими категориями пользователей

Прежде всего, если банк данных создается для информационного обслуживания какого-либо предприятия или организации, необходимы контакты с администрацией этой организации. Как указывалось выше, внедрение БнД приводит к большим изменениям не только системы обработки данных, но и всей системы управления организацией. Естественно, что такие большие проекты не могут быть выполнены без активного участия и поддержки руководителей организации. Руководство организации должно быть ознакомлено с возможностями, предоставляемыми БнД, проинформировано об их преимуществах и недостатках, а также о проблемах, вызываемых созданием и функционированием БнД.

Поскольку база данных является динамическим информационным отображением предметной области, то желательно, чтобы администратор БнД в свою очередь был своевременно информирован о перспективах развития объекта, для которого создается информационная система.

Руководством организации и администратором БнД должны быть согласованы цели, основные направления и сроки создания БнД и его развития, очередность подключения пользователей.

Очень тесная связь у АБД на всех этапах жизненного цикла БнД наблюдается с конечными пользователями. Это взаимодействие возникает на начальных стадиях проектирования системы, когда изучаются потребности пользователей, уточняются особенности предметной области, и постоянно поддерживается в процессе проектирования и функционирования системы.

Следует отметить, что в последнее время наблюдается активное перераспределение функций между конечными пользователями и администраторами банка данных. Это, прежде всего, связано с развитием языковых и программных средств, ориентированных на конечных пользователей. К ним относятся простые и одновременно мощные языки запросов, а также средства автоматизации проектирования.

Если банк данных функционирует в составе какой-либо автоматизированной информационной системы (например, в АСУ), то АБД должен работать в контакте со специалистами по обработке данных в этой системе.

Администраторы БнД взаимодействуют также с внешними по отношению к нему группами специалистов, и, прежде всего поставщиками СУБД и ППП, администраторами других БнД.

БнД часто создаются специализированными проектными коллективами на основе договора на разработку информационной системы в целом или БнД как самостоятельного объекта проектирования. В этом случае служба администрации БнД должна создаваться как в организации-разработчике, так и в организации-заказчике.

Средства администратора современных СУБД. На эффективность работы БнД оказывает влияние множество внешних и внутренних факторов. Возрастание сложности и масштабов БнД, высокая цена неправильных или запоздалых решений по администрированию БД, высокие требования к квалификации специалистов делают актуальной задачу использования развитых средств автоматизированного (или даже автоматического) администрирования БнД.

Средства администрирования включены в состав всех СУБД. Особенно развиты эти средства в корпоративных СУБД. Кроме того, появился целый класс специализированного программного обеспечения - средства DBA (DataBase Administration - администрирование базы данных).

Типичными функциями средств DBA являются:

1. Мониторинг работы БД, реакция на нештатные ситуации:
   - слежение за использованием ресурсов, выдача статистики;
   - обнаружение и исправление возникающих неполадок.
2. Наблюдение за объектами БД, анализ, сопоставление характеристик:
   - планирование необходимых вычислительных мощностей;
   - задание пороговых значений для слежения за нужными объектами.
3. Оптимизация хранения данных, оптимизация работы сервера:

- анализ свободного пространства, устранение дефрагментации;
- наблюдение за параметрами, влияющими на производительность БнД.

4. Сопровождение БД, файлов, табличных пространств, откатных сегментов:

- перенос таблицы на новое пространство, в другую СУБД, на другой компьютер;
- перенос содержимого базы данных в другую СУБД.

1.2.7. Взаимодействие компонентов БнД

На рис. 1.12 представлена упрощенная схема взаимодействия компонентов БнД в процессе создания и эксплуатации системы. Создание БД начинается с проектирования БД и ее описания на ЯОД (1). На этапе проектирования структуры БД могут использоваться как методики ручного проектирования, так и CASE-средства, автоматически генерирующие описания БД. Полученные описания должны быть введены в БнД и запомнены в соответствии с требованиями конкретной СУБД (2,3). После того как описание базы данных сохранено, в базу данных могут вводиться данные (4). При этом СУБД использует метаинформацию, зафиксированную в словаре данных. Заполненная БД может использоваться для извлечения из нее нужной пользователям информации (5). При формулировании запросов используется информация, содержащаяся в схемах и подсхемах. В результате выполнения запроса выходные данные в том или ином виде выдаются пользователю (6). Кроме собственно затребованных данных при выполнении операций с БнД часто выдается диагностическая информация (7). Для обеспечения надежности функционирования БнД необходимо выполнять соответствующие процедуры, в частности осуществлять журнализацию (8) выполняемых действий с БД, регулярно архивировать данные (9).

1.3. Классификация банков данных

Банки данных являются сложными системами, и их классификация может быть произведена как для всего банка данных в целом, так и для каждого его компонента отдельно; классификация для каждого компонента может быть проведена по множеству разных признаков (рис. 1.13).

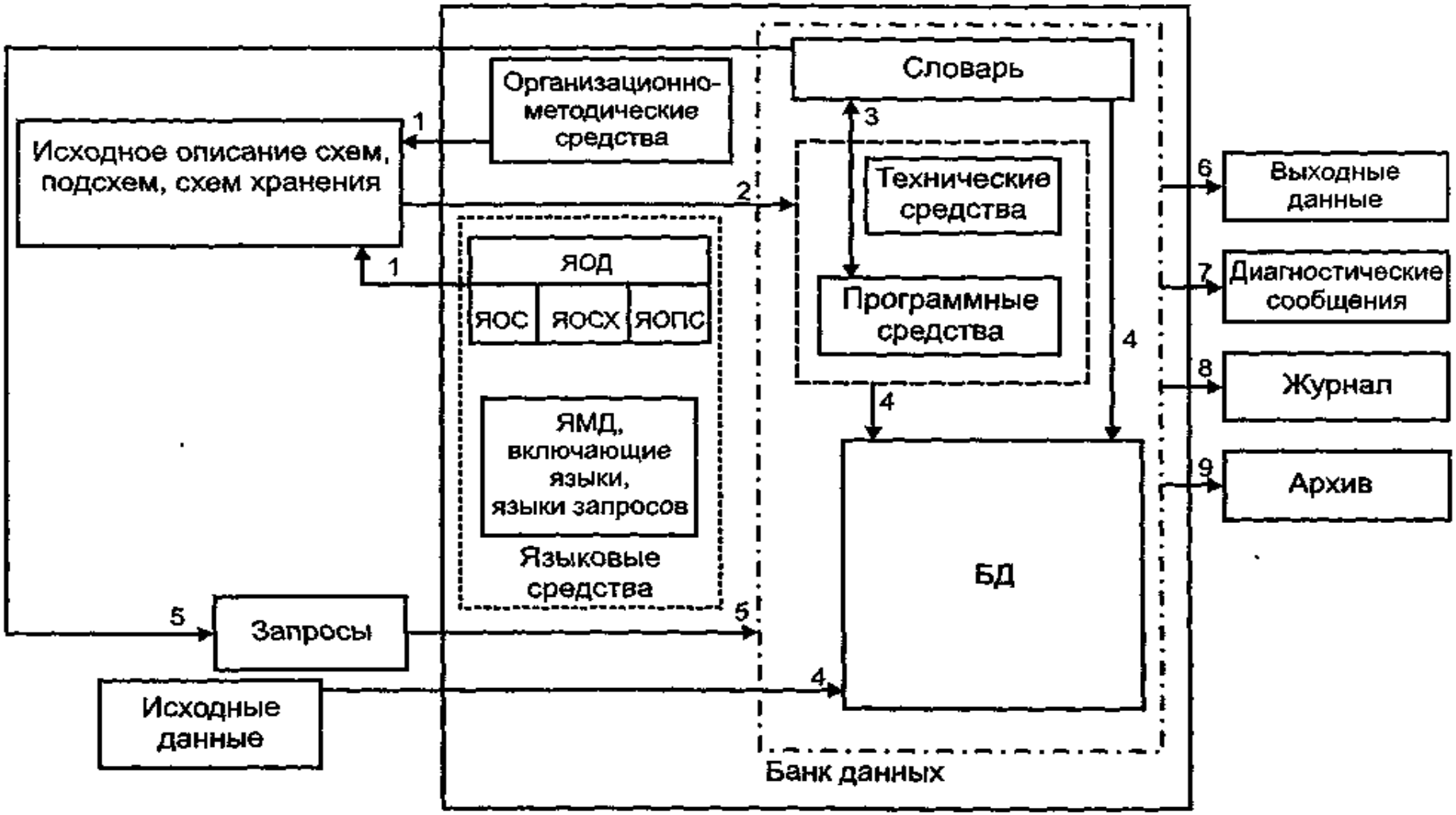


Рис. 1.12. Схема взаимодействия компонентов БнД

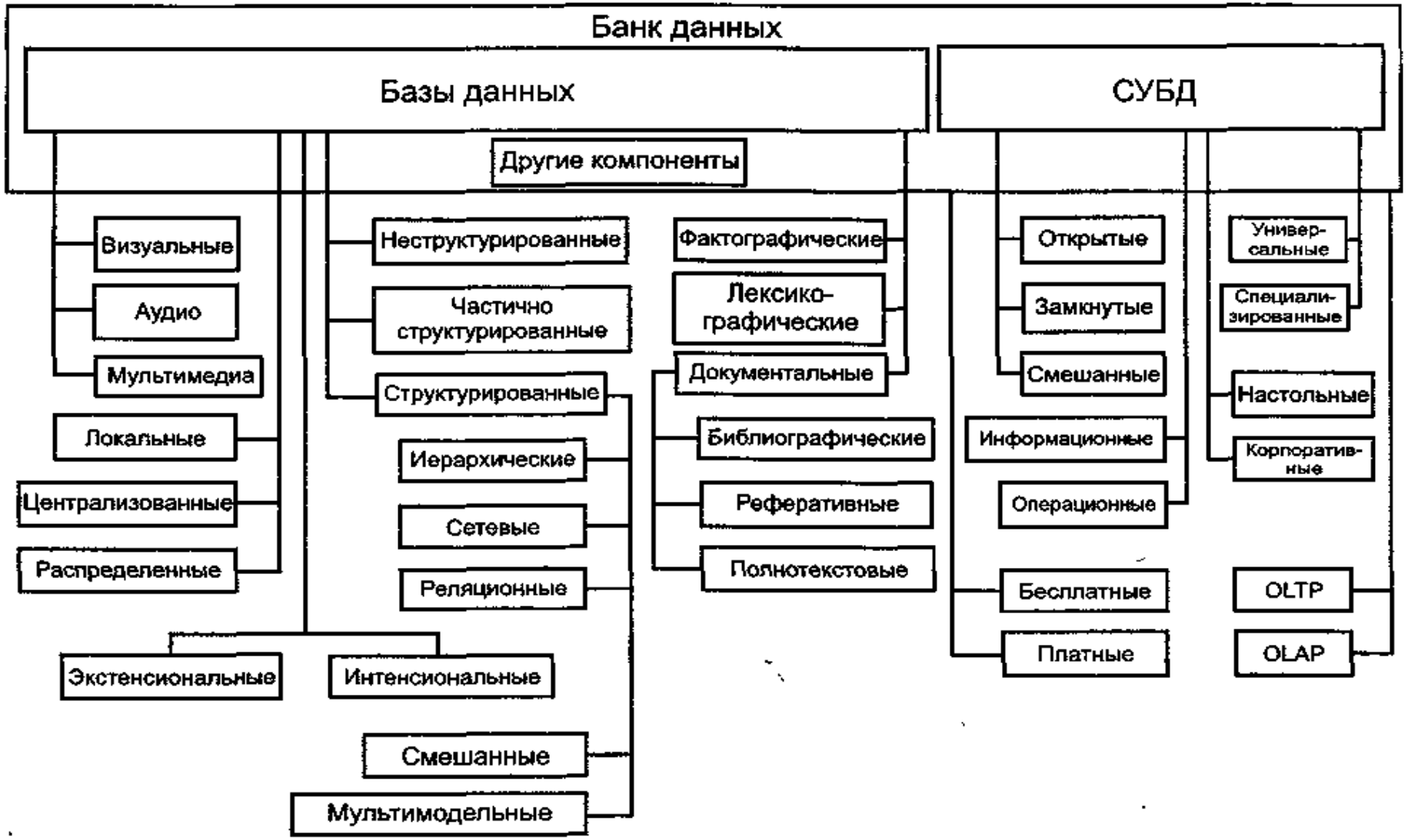


Рис. 1.13. Классификация БнД

1.3.1. Классификация баз данных

Центральным компонентом банка данных является база данных, и большинство классификационных признаков относятся именно к ней. По *форме представления информации* различают визуальные и аудиосистемы, а также системы мультимедиа. Эта классификация показывает, в каком виде информация хранится в БД и выдается из баз данных пользователям - в виде изображения, звука или имеется возможность использования разных форм отображения информации. Понятие «изображение» здесь используется в широком смысле. Это может быть символьный текст, неподвижное графическое изображение (рисунки, чертежи и т.п.), фотографии, географические карты, движущиеся изображения. Классификация способов представления информации являет собой самостоятельную проблему и здесь не рассматривается.

По *характеру организации данных* БД могут быть разделены на неструктурированные, частично структурированные и структурированные. Этот классификационный признак относится к информации, представленной в символьном виде. К неструктурированным БД могут быть отнесены базы, организованные в виде семантических сетей. Частично структурированными можно считать базы данных в виде обычного текста или гипертекстовые системы. Структурированные БД требуют предварительного проектирования и описания структуры БД. Только после этого базы данных такого типа могут быть заполнены данными.

Структурированные БД, в свою очередь, по *типу используемой модели* делятся на иерархические, сетевые, реляционные, смешанные и мультимодельные.

Классификация по типу модели распространяется не только на базы данных, но и на СУБД.

В структурированных БД обычно различают несколько уровней информационных единиц (ИЕ), входящих одна в другую. Число этих уровней может быть различным даже для систем, относящихся к одному и тому же классу. Большинство структурированных систем поддерживают уровень поля, записи и файла. Эти информационные единицы могут называться в разных системах по-разному, но суть остается одной и той же, а именно: *полю* соответствует наименьшая семантическая единица информации; совокупность полей или иных, более сложных информационных единиц, если они допустимы в конкретной СУБД, образует *запись,* а множество однотипных записей представляет *файл базы данных.* В последнее время большинство СУБД в явном виде поддерживают и уровень базы данных как совокупности взаимосвязанных файлов БД.

В иерархических (рис. 1.14) и сетевых (рис. 1.15) моделях между информационными единицами (записями разных файлов) могут задаваться связи. Как видно из приведенных схем, графическое представление *иерархической модели* представляет собой граф типа «дерево». В такой модели имеется одна вершина - корень дерева, являющаяся входом в структуру. Каждая вер-

шина, отличная от корня, может иметь только одну исходную вершину и в общем случае сколько угодно порожденных вершин.

Графическое представление *сетевой модели* представляет собой граф типа «сеть». Входом в такую структуру может являться любая вершина. Каждая вершина может иметь как несколько порожденных, так и несколько исходных вершин. Между парой вершин может быть объявлено несколько связей. Подавляющее большинство СУБД поддерживает простые сетевые структуры, т.е. между каждой парой типов записей поддерживается отношение 1:M (один ко многим). Направление и характер связи в сетевых моделях не являются очевидными, как в случае иерархической модели, поэтому при изображении структуры БД направление связи должно быть указано.

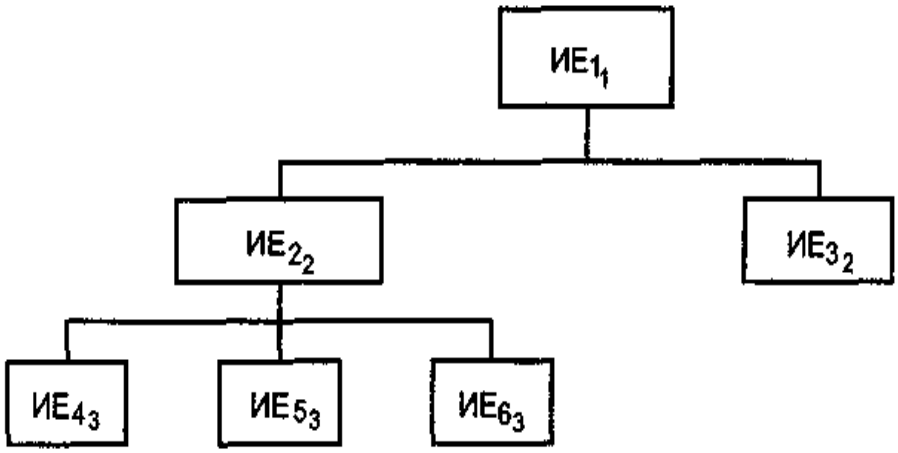


Рис. 1.14. Схема иерархической модели

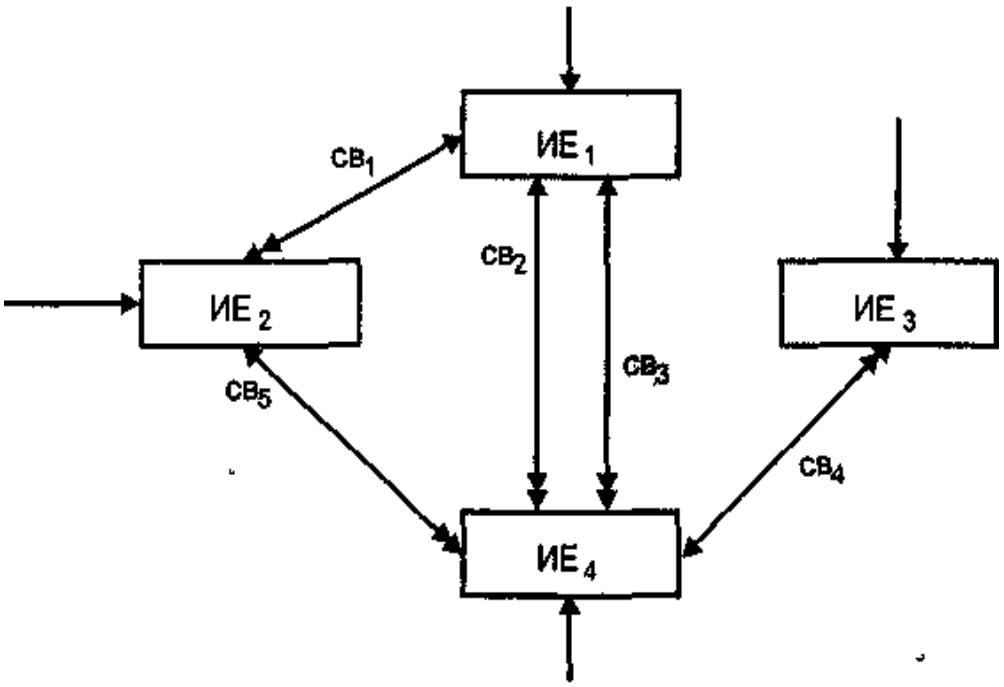


Рис. 1.15. Схема сетевой модели с однотипными файлами

В сетевой модели с однотипными файлами *(см.* рис. 1.15) каждый файл может служить входом в структуру. Пара связанных файлов называется *набором.* В наборе тот файл, от которого идет связь, называется *владельцем* набора, а файл, к которому направлена эта связь, - *членом* набора. Файл, который в одном наборе является членом, в другом может быть владельцем. Другими словами, тип файла жестко не зафиксирован.

Кроме сетевых моделей с однотипными файлами существуют сетевые модели с разнотипными файлами. В таких моделях различают главные (основные) и зависимые файлы (рис. 1.16). Вход в структуру возможен только через главные файлы. Связываться между собой могут только записи разных типов.

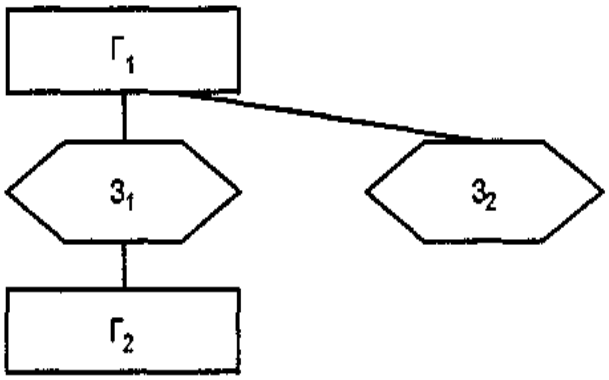


Рис. 1.16. Схема сетевой модели с разнотипными файлами: Г - главный файл; 3 – зависимый

Связи в иерархических и сетевых моделях описываются при проектировании БД. Чаще всего эти связи при хранении данных в БД передаются посредством адресных указателей. Иерархические и сетевые модели БД не накладывают ограничений на тип внутризаписной структуры. В принципе она может быть любой: как простой линейной (т.е. состоять только из простых полей, следующих в записи последовательно друг за другом), так и сложной иерархической, включающей в себя различные составные единицы информации (векторы, повторяющиеся группы и т.п.). Конкретные же СУБД накладывают ограничения на допустимые в них информационные единицы, характер связей между ними, порядок их расположения в записи, а также часто имеют и различные количественные ограничения.

Отдельное место среди структурированных систем занимают системы, построенные на использовании *инвертированных файлов.* Особенность организации данных в них состоит в том, что собственно хранимые данные и информация о связях между информационными единицами логически и физически отделены друг от друга. Основные данные в этих системах хранятся в файлах, записи которых могут иметь сложную структуру. Вся управляющая информация сосредоточена в ассоциаторе. Логическая связь между файлами устанавливается посредством компонента ассоциатора, называемого сетью связи. На рис. 1.17 схематически представлен принцип установления связей в таких системах. Реально связи устанавливаются не непосредственно с эле-

ментами связи, как это изображено на рисунке, а через преобразователь адреса. В системах, построенных на инвертированных файлах, можно передавать связь типа М: М (многие ко многим) между записями файлов, что не позволяют никакие другие системы. Отделение ассоциативной информации от собственно хранимых данных позволяет изменять связи, не изменяя при этом самих файлов.

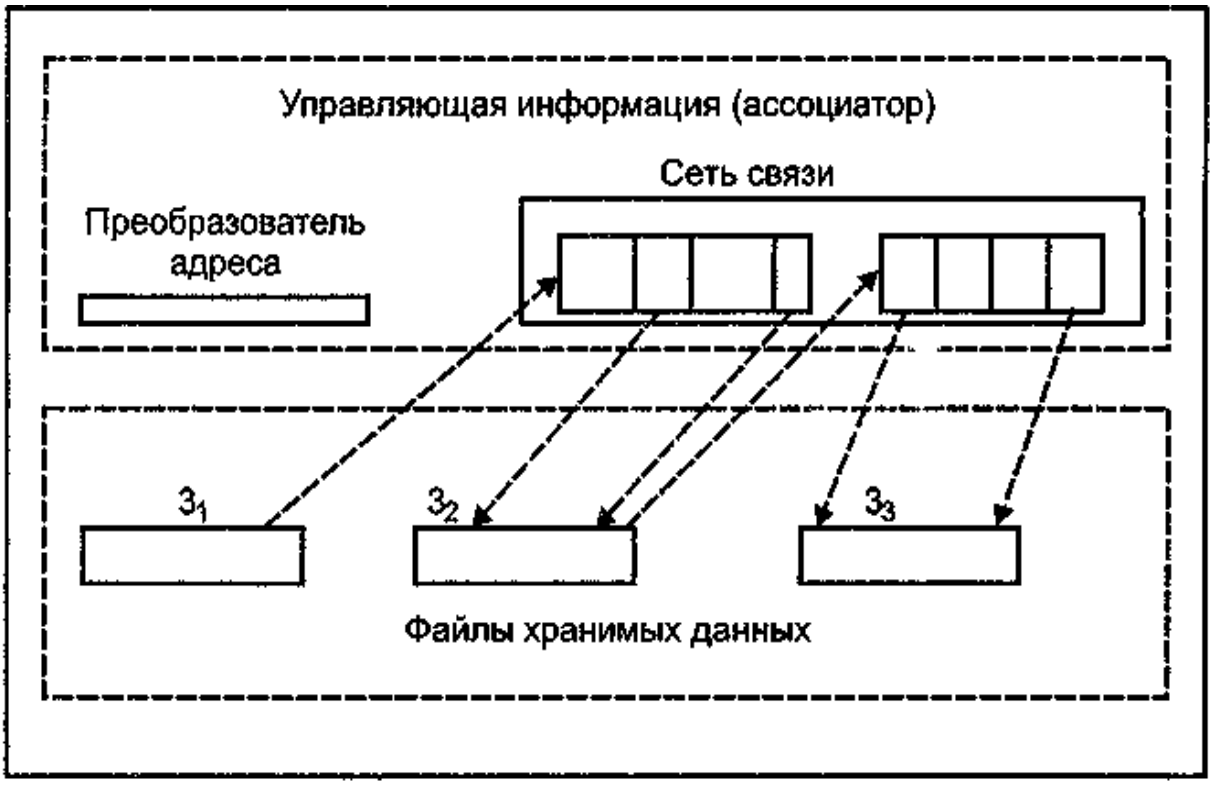


Рис. 1.17. Схема организации данных в системах, основанных на инвертированных файлах

В *реляционных* моделях (в отличие от иерархических и сетевых) связи между записями разных таблиц БД определяются динамически в момент выполнения запроса. Эти связи устанавливаются по равенству значений соответствующих полей (полей связи), содержащихся в каждом из связанных файлов/таблиц (рис. 1.18).

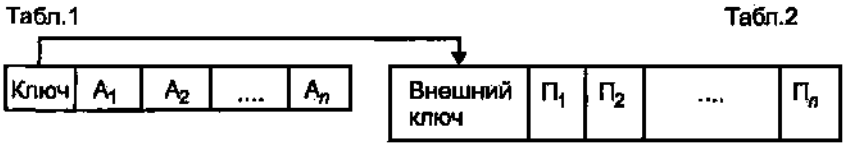


Рис. 1.18. Схема реляционной модели:
A_n, $П_n$ — имена полей

Другой отличительной чертой реляционных моделей является ограничение на внутризаписную структуру: записи имеют линейную структуру и мо-

гут содержать только простые поля. Эти особенности играют решающую роль при проектировании структуры БД.

Третьей отличительной особенностью реляционных моделей является использование теоретико-множественных языковых средств: реляционной алгебры или реляционного исчисления.

1980-е гг. были временем интенсивного развития реляционных систем. В 1992 г*. уровень продаж реляционных СУБД впервые превысил уровень продаж нереляционных СУБД. Но до 90% данных предприятий хранилось к этому моменту в нереляционных базах данных на мейнфреймах.

В связи с тем, что в настоящее время основной сегмент рынка занят именно реляционными СУБД, в дальнейшем главное внимание уделим этим системам. В приложении 1 даны основные понятия теории реляционных баз данных, знание которых необходимо при работе **с** ними.

По ***типу хранимой информации*** БД делятся на документальные, фактографические и лексикографические. Среди *документальных* баз различают библиографические, реферативные и полнотекстовые.

К *лексикографическим* базам данных относятся различные словари (классификаторы, многоязычные словари, словари основ слов и т. п.).

В системах *фактографического* типа в БД хранится информация об интересующих пользователя объектах предметной области в виде «фактов» (например, биографические данные о сотрудниках, данные о выпуске продукции производителями и т.п.); в ответ на запрос пользователя выдается требуемая информация об интересующем его объекте (объектах) или сообщение о том, что искомая информация отсутствует в БД.

В документальных БД единицей хранения является какой-либо документ (например, текст закона или статьи), и пользователю в ответ на его запрос выдается либо ссылка на документ, либо сам документ, в котором он может найти интересующую его информацию.

БД документального типа могут быть организованы по-разному: без хранения и с хранением самого исходного документа на машинных носителях. К системам первого типа можно отнести библиографические и реферативные БД, а также БД-указатели, отсылающие к источнику информации. Системы, в которых предусмотрено хранение полного текста документа, называются *полнотекстовыми*.

В системах документального типа целью поиска может быть не только какая-то информация, хранящаяся в документах, но и сами документы. Так, возможны запросы типа «сколько документов было создано за определенный период времени» и т. п. Часто в критерий поиска в качестве признаков включаются «дата принятия документа», «кем принят» **и** другие «выходные данные» документов.

Специфической разновидностью баз данных являются базы данных форм документов. Они обладают некоторыми чертами документальных сис-

---

* Первая коммерческая реляционная СУБД была выпущена фирмой Oracle в 1979 г.

тем (ищется документ, а не информация о конкретном объекте, форма документа имеет название, по которому обычно и осуществляется его поиск), и специфическими особенностями (документ ищется не с целью извлечь из него информацию, а с целью использовать его в качестве шаблона).

В последние годы активно развивается объектно-ориентированный подход к созданию информационных систем. Объектные базы данных организованы как объекты и ссылки к объектам. Объект представляет собой данные и правила, по которым осуществляются операции с этими данными. Объект включает метод, который является частью определения объекта и запоминается вместе с объектом. В объектных базах данных данные запоминаются как объекты, классифицированные по типам классов и организованные в иерархическое семейство классов. *Класс* - коллекция объектов с одинаковыми свойствами. Объекты принадлежат классу. Классы организованы в иерархии.

По ***характеру организации хранения данных и обращения к ним*** различают локальные (персональные), общие (интегрированные, централизованные) и распределенные базы данных (рис. 1.19).

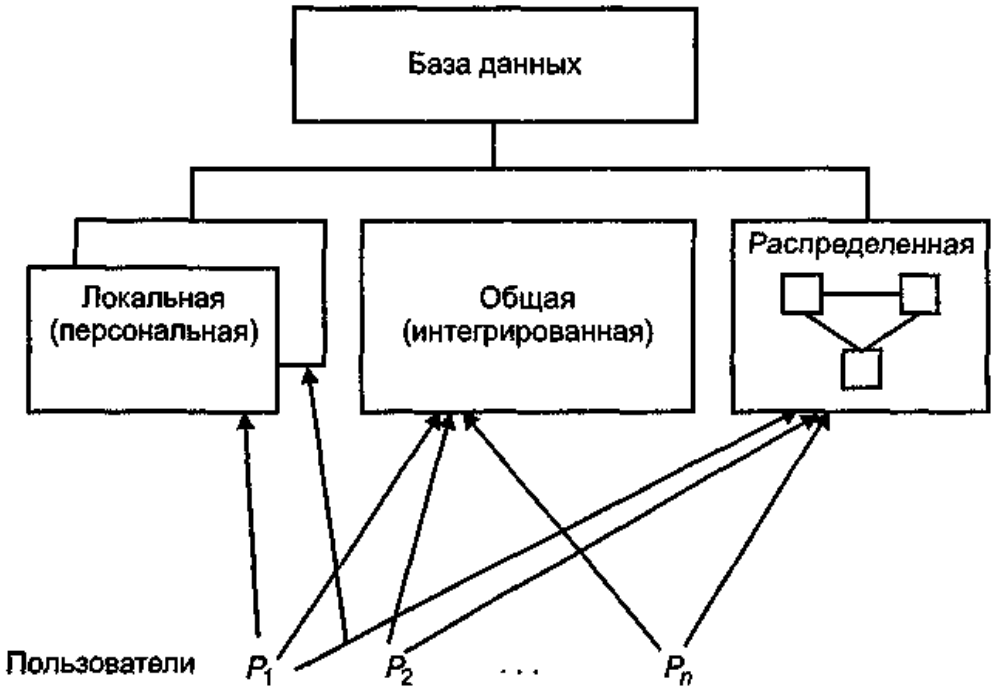


Рис. 1.19. Классификация базы данных по характеру хранения данных и обращения к ним

Персональная база данных - это база данных, предназначенная для локального использования одним пользователем. Локальные БД могут создаваться каждым пользователем самостоятельно, а могут извлекаться из общей БД.

Интегрированные и *распределенные БД* предполагают возможность одновременного обращения нескольких пользователей к одной и той же информации (многопользовательский, параллельный режим доступа). Это

привносит специфические проблемы при их проектировании и в процессе эксплуатации БнД. Распределенные БД, кроме того, имеют характерные особенности, связанные с тем, что физически разные части БД могут быть расположены на разных ЭВМ, а логически, с точки зрения пользователя, они должны представлять собой единое целое.

Технологии, которые, на первый взгляд, вроде бы находятся на разных концах спектра (локальная и распределенная обработка), на самом деле очень близки и различаются практически тем, как поддерживается связь между отдельными частями БД. В локальных системах поддержание этой связи не является централизованным, а в распределенных БнД связь должна поддерживаться СУБД. Совмещать идеи локальной работы и централизованного поддержания единой БД позволяет технология тиражирования, при которой средства СУБД обеспечивают тиражирование отдельных частей общей БД, локальное использование их, а затем согласование отдельных фрагментов БД в рамках единой базы данных.

Концепции централизованной и распределенной обработки данных гоже не так сильно различаются между собой, как кажется на первый взгляд. Так называемые клиент-серверные системы с тонким клиентом очень близки к централизованным базам данных.

Банк данных является сложной человеко-машинной системой, и распределяться по узлам сети могут не только БД, но и другие компоненты БнД. Причем сама БД при этом может быть и нераспределенной (например, при обеспечении многопользовательского доступа к централизованной БД в сети). Поэтому будем различать два понятия: распределенные БД и распределенные БнД. При этом под *распределенным БнД* будем понимать банк данных, в котором распределен хотя бы один его компонент.

Распределенные БнД, в свою очередь, могут быть классифицированы по разным признакам. Дополнительная информация о распределенных БнД будет дана в главе 10.

В [39] различают экстенсиональные (ЭБД) и интенсиональные (ИБД) базы данных. Последние строятся с помощью правил, определяющих их содержание, а не с помощью явного хранения данных в БД, как экстенсиональные БД.

Например, пусть имеется ЭБД, содержащая таблицу «Личность (Person)», которая содержит сведения о личности и среди полей которой есть поля «Фамилия_имя_отчество (FIO)», «Пол (Sex)». Можно построить в этой ЭБД вторую таблицу «Родитель (Parent)», которая содержит поля «Фамилия_имя_отчество» родителя (FIO) и «Имя_ребенка (Child)». С помощью правил можно определить, например, отношение «Отец (Father)», просто указав, что отец - это родитель, у которого пол - мужской. На ПРОЛОГЕ это отношение можно определить следующим образом:

father(X,Y):= person(X,male), parent(X,Y).

Если выполнить это правило, то получится отношение, которое содержит подмножество кортежей таблицы «Parent», таких, для которых верно указанное условие. Пользователю эти данные будут выданы в виде обычного отношения.

Данное определение ЭБД и ИБД можно расширить и на другой (не реляционный) тип БД, и на другой способ задания правил. В более общем виде можно сказать, что информацию можно передать и в виде данных, и в виде программ (строго говоря, программы тоже являются данными, но в русском языке нет подходящего термина, который можно было бы здесь употребить вместо слова «данные»).

БД классифицируются по ***объему***. Особое место здесь занимают так называемые *очень большие базы данных.* Это вызвано тем, что для больших баз данных по-иному ставятся вопросы обеспечения эффективности хранения информации и обеспечения ее обработки.

1.3.2. Классификации СУБД

Рассмотрим теперь ряд классификационных признаков, относящихся к СУБД. По ***языкам общения*** СУБД делятся на открытые, замкнутые и смешанные. *Открытые системы* - это системы, в которых для обращения к базам данных используются универсальные языки программирования. *Замкнутые системы* имеют собственные языки общения с пользователями БнД.

По ***числу уровней*** в архитектуре различают одноуровневые, двухуровневые, трехуровневые системы. В принципе возможно выделение и большего числа уровней. Под *архитектурным уровнем СУБД* понимают функциональный компонент, механизмы которого служат для поддержки некоторого уровня абстракции данных (логический и физический уровень, а также «взгляд» пользователя - внешний уровень). Более подробно о каждом из названных уровней *см.* в разд. 1.5.1.

На рис. 1.20 сделана попытка совместить терминологию, встречающуюся в разных литературных источниках. В литературе широко используются понятия «внешняя», «концептуальная» и «внутренняя» модель/уровень *(см.,* например, [28] и др.), «логический» и «физический»* уровень [39], а также «внешняя схема», «подсхема», «схема хранения», просто «схема» и пр. Понятие «схема» с тем или иным уточнением обычно относится к описанию соответствующего уровня описания данных.

Нумерация уровней на рисунке условна, но, тем не менее, отражает их значимость (внутренняя модель может быть построена только на основе кон-

* Во многих современных CASE-средствах концептуальной моделью называется ER-модель предметной области *(см.* главу 2), а физической - модель, поддерживаемая конкретной СУБД. Если первое еще можно считать удачным использованием термина (так как ER-модель действительно отражает общую «концепцию» системы), то второе крайне неудачно, поскольку ни о какой «физике» речь здесь не идет.

цептуальной; эти два уровня могут быть совмещены, но поддерживаются СУБД всегда; внешний уровень в архитектуре СУБД может отсутствовать).

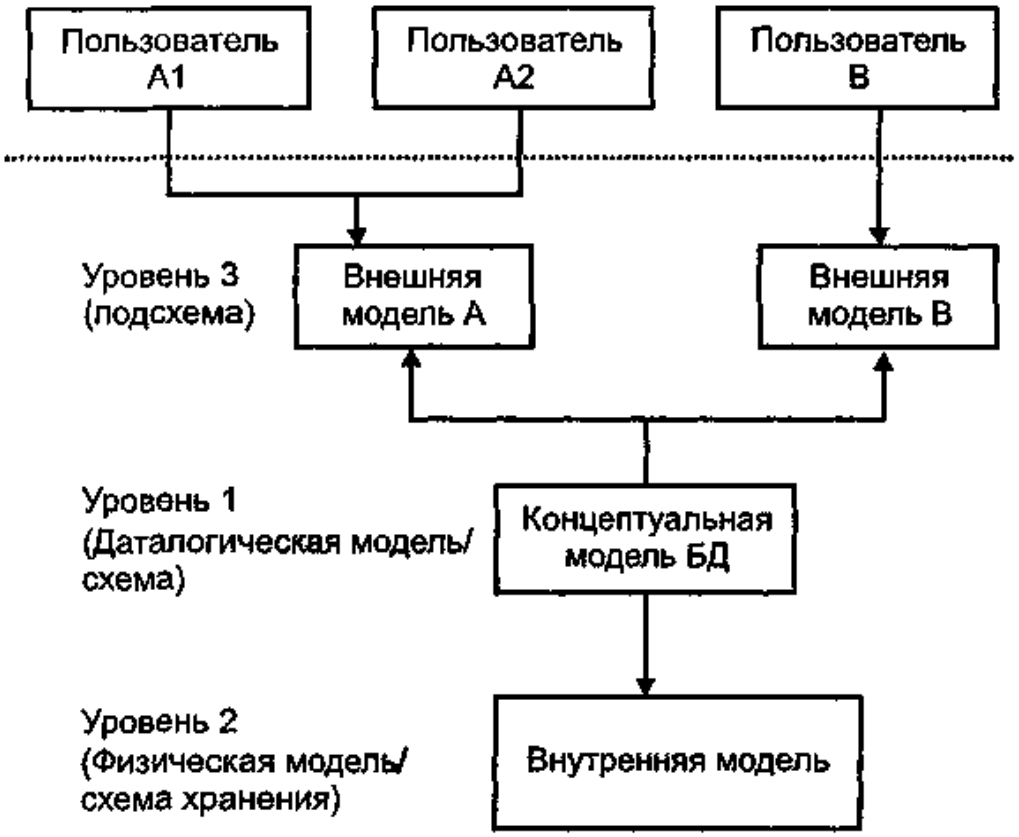


Рис. 1.20. Классификация СУБД по числу уровней в архитектуре (пример трехуровневой архитектуры)

По ***выполняемым функциям*** СУБД делятся на информационные и операционные. *Информационные СУБД* позволяют организовать хранение информации и доступ к ней. Для выполнения более сложной обработки необходимо писать специальные программы. *Операционные СУБД* выполняют достаточно сложную обработку, например, автоматически позволяют получать агрегированные показатели, не хранящиеся непосредственно в базе данных, могут изменять алгоритмы обработки и т.д.

По ***сфере возможного применения*** различают универсальные и специализированные, обычно проблемно-ориентированные СУБД.

Системы управления базами данных поддерживают разные типы данных. Набор типов данных, допустимых в разных СУБД, различен. Кроме того, ряд СУБД позволяет разработчику добавлять новые типы данных и новые операции над этими данными. Такие системы называются *расширяемыми* системами баз данных (РСБД).

Дальнейшим развитием концепции РСБД являются *системы объектно-ориентированных баз данных (СООБД),* обладающие достаточно мощными

выразительными возможностями, чтобы непосредственно моделировать сложные объекты.

По ***мощности*** СУБД делятся на настольные и корпоративные. Характерными чертами *настольных СУБД* являются сравнительно невысокие требования к техническим средствам, ориентация на конечного пользователя, низкая стоимость.

Корпоративные СУБД обеспечивают работу в распределенной среде, высокую производительность, поддержку коллективной работы при проектировании систем, имеют развитые средства администрирования и более широкие возможности поддержания целостности. Эти системы сложны, дороги, требуют значительных вычислительных ресурсов.

Сравнительные характеристики настольных и корпоративных СУБД приведены в табл. 1.1.

Таблица 1.1

Критерий	Настольные	Корпоративные
Простота использования	+	
Стоимость программного обеспечения	+	
Стоимость эксплуатации	+	
Функциональные возможности: администрирование, работа с Интернет/интранет и др.		+
Надежность функционирования		+
Поддерживаемые объемы данных		+
Быстродействие		+
Возможности масштабирования		+
Работа в гетерогенной среде		+

Системы обоих классов интенсивно развиваются, причем некоторые тенденции развития присущи каждому из этих классов. Прежде всего, это использование высокоуровневых средств разработки приложений (что раньше было присуще в основном настольным системам), рост производительности и функциональных возможностей, работа в локальных и глобальных сетях и др.

Наиболее известными из корпоративных СУБД являются Oracle, DB2, Sybase, MS SQL Server, Progress и некоторые другие.

Наблюдается связь между классом СУБД и используемой операционной системой. Системы под UNIX позиционируются как корпоративные распределенные системы. Сейчас в этот сектор «пробивается» Windows NT и заменяющая ее Windows 2000.

По ***ориентации на преобладающую категорию пользователей*** можно выделить СУБД *для разработчиков* и *для конечных пользователей.* Системы, относящиеся к первому классу, должны иметь качественные компиляторы и позволять создавать «отчуждаемые» программные продукты, обладать развитыми средствами отладки, включать средства документирования проекта и

другие возможности, позволяющие строить эффективные сложные системы. Основными требованиями, предъявляемыми к системам, ориентированным на конечного пользователя, являются: удобство интерфейса, высокий уровень языковых средств, наличие интеллектуальных модулей подсказок, повышенная защита от непреднамеренных ошибок («защита от дурака») и т.п.

Существует разделение СУБД ***по поколениям***. К первому поколению СУБД относят системы, основанные на иерархической и сетевой моделях (60-70-е гг. XX в.), ко второму поколению - реляционные системы. СУБД третьего поколения, требования к которым были сформулированы в «Манифесте систем баз данных третьего поколения» [38], должны поддерживать сложные структуры данных и более развитые средства обеспечения целостности данных, отвечать требованиям, предъявляемым к открытым системам.

1.3.3. Классификационные группировки, относящиеся к БнД в целом

Следующая группа признаков классификации связана с банком данных в целом. По ***условиям предоставления услуг*** различают *бесплатные* и *платные* банки данных. Платные БнД, в свою очередь, делятся на *бесприбыльные* и *коммерческие.* Бесприбыльные банки данных функционируют на принципе самоокупаемости и не ставят своей целью получение прибыли. Это обычно БнД социально значимой информации, имеющей широкий круг пользователей, или научной, библиотечной информации. Основной целью создания коммерческих банков данных является получение прибыли от информационной деятельности.

Информационные системы различаются ***по характеру преобладающей обработки информации.*** **В** одних в основном реализуется большое число достаточно простых запросов. Такие системы получили название *OLTP* (On-Line Transaction Processing - *системы оперативной обработки транзакций).* В других, напротив, требуется сложная аналитическая обработка данных. Для такого класса систем стал использоваться термин *OLAP* (On-Line Analytical Processing).

Термин OLAP - сравнительно новый и в разных литературных источниках трактуется иногда по-разному. Его часто отождествляют с поддержкой принятия решений (DSS (Decision Support Systems) – *системы поддержки принятия решения).* А в качестве синонима для последнего термина используют Data Warehousing - хранилища (склады) данных, понимая под этим набор организационных решений, программных и аппаратных средств для обеспечения аналитиков информацией на основе данных из систем обработки транзакций нижнего уровня и других источников.

«Склады данных» позволяют обрабатывать данные, накопленные за длительные периоды времени. Эти данные являются разнородными (и не обязательно структурированными). Для «складов данных» присущ многомерный характер запросов. Огромные объемы данных, сложность структуры, как данных, так и запросов требуют использования специальных методов доступа к информации.

В других источниках понятие системы поддержки принятия решений (СППР) считается более широким. Хранилища данных и средства оперативной аналитической обработки могут служить компонентами архитектуры СППР.

OLAP «в узком смысле» - это системы, которые обеспечивают только выборку данных в различных разрезах. OLAP «в широком смысле», или просто OLAP, включают в себя:

- поддержку нескольких пользователей, редактирующих БД;
- функции моделирования, в том числе вычислительные механизмы получения производных результатов, а также агрегирования и объединения данных;
- прогнозирование, выявление тенденций и статистический анализ.

Естественно, что каждый из этих типов ИС требует специфической организации данных, а также специальных программных средств, обеспечивающих эффективное выполнение стоящих задач.

Для обеспечения быстрой обработки данных при их анализе используются разнообразные приемы. Одним из них является организация данных в виде так называемых многомерных БД (MDD -MultiDimensional Database). Информация в MDD хранится не в виде индексированных записей в таблицах, а в форме логически упорядоченных массивов. Единой общепризнанной многомерной модели храпения данных не существует. В MDD отсутствует стандартизованный метод доступа к данным, и они могут отвечать требованиям специфической аналитической обработки данных.

Сравнительные характеристики систем OLTP и OLAP приведены в табл. 1.2.

Таблица 1.2

Характеристика	OLTP	OLAP
Преобладающие операции	Ввод данных, поиск	Анализ данных
Характер запросов	Много простых транзакций	Сложные транзакции
Хранимые данные	Оперативные, детализированные	Охватывающие большой период времени, агрегированные
Вид деятельности	Оперативная, тактическая	Аналитическая, стратегическая
Тип данных	Структурированные	Разнотипные

Хранилища данных могут быть разбиты на два типа: корпоративные хранилища данных (enterprise data warehouses) и киоски данных (data marts).

Корпоративные хранилища данных содержат информацию, относящуюся ко всей корпорации и собранную из множества оперативных источников для консолидированного анализа. Обычно такие хранилища охватывают це-

лый ряд аспектов деятельности корпорации и используются для принятия как тактических, так и стратегических решений.

Киоски данных содержат подмножество корпоративных данных и строятся для отделов или подразделений внутри организации. Киоски данных часто строятся силами самого отдела и охватывают конкретный аспект, интересующий сотрудников данного отдела. Киоск данных может получать данные из корпоративного хранилища (зависимый киоск), или, что более распространено, данные могут поступать непосредственно из оперативных источников (независимый киоск).

Киоски и хранилища данных строятся по сходным принципам и используют практически одни и те же технологии.

По ***степени доступности*** БнД делятся на *общедоступные* и *с ограниченным кругом пользователей.*

По ***охвату*** БД могут классифицироваться в разных разрезах:

- территориальный:
  всемирный;
  страна;
  город;
- временной;
- ведомственный;
- проблемный (тематический).

Территориальный и ведомственный признаки классификации могут относиться не только к информации, хранящейся БД, но и к кругу обслуживаемых пользователей.

По ***характеру взаимодействия с пользователями*** (кто инициализирует действия) БнД делятся на активные и пассивные.

В *пассивных* БнД ведущая роль принадлежит пользователю, в *активных* - система может самостоятельно менять поведение. В последнее время термин «активная база данных» стал часто использоваться для систем, использующих триггеры.

По ***форме собственности*** БнД делятся на государственные и негосударственные (частные, групповые, личные).

В литературе встречаются и другие аспекты классификации банков данных, но названные являются наиболее значимыми.

1.4. Выбор СУБД

1.4.1. Тенденции развития СУБД

Рынок СУБД бурно развивается, достаточно широк и разнообразен. Тем не менее, можно выделить некоторые основные тенденции, присущие этому классу программных продуктов.

1. Поддержка разных стилей проектирования (традиционный, объектно-ориентированный, «визуальное» проектирование).

Использование визуальных объектно-ориентированных средств разработки является преобладающей тенденцией.

2. Использование мультимедийных систем. Обработка графических образов. Управление пространственной информацией.

3. Повышение уровня языковых средств. Дружелюбность.

4. Включение в СУБД разнообразных языковых средств, ориентированных на разные категории пользователей.

5. Поддержка решений для реализации хранилищ данных в целях обеспечения процессов принятия решений.

6. Наличие разнообразных сервисных средств (развитая «помощь»: контекстная помощь (Help), электронные учебники, примеры, шаблоны распространенных приложений, «Волшебники» и т.п.).

7. Расширение функциональных возможностей (многофункциональность). Причем расширяется набор, и улучшаются характеристики не только функций, присущих собственно СУБД, но и существенно развиваются возможности получения выходных документов различных видов, функции редакторов (в том числе проверка орфографии и т.п.), передача документов по ЭП, документирование проекта и некоторые другие функции, которые не являются функциями по управлению данными.

8. Многоплатформенность - наличие вариантов «одноименных» СУБД, реализованных для разных операционных систем и разных типов компьютеров. Несмотря на острую конкуренцию операционных систем, иногда бывают периоды бурного развития и широкого распространения какой-то одной из них. Так, например, 1990-е гг. характеризовались массовым выпуском СУБД под Windows для настольных систем. Для корпоративных систем пальму первенства долгое время удерживала UNIX.

9. Рост производительности. Он присущ системам всех классов, но, разумеется, сама производительность для каждого класса систем различается. Наибольшей производительностью обладают СУБД, относящиеся к классу корпоративных систем.

10. Использование более развитых и разнообразных средств обеспечения целостности и безопасности данных.

11. «Интернационализация». Выражается в использовании СУБД ведущих производителей в разных странах мира; со стороны производителя это проявляется в локализациях версий. Поддержка национальных языков (Спецификация NLS - National Language Support).

12. Работа в многопользовательской среде. Распределенность. Работа в гетерогенной среде. Масштабируемость.

13. Открытость системы. Обеспечивается поддержкой определенных стандартов.

14. Преобладающая модель данных - реляционная. Наблюдается повышение «уровня реляционности» данных. Например, многие системы стали поддерживать концепции ключа, вероятного ключа, ссылочной целостности. Само понятие БД, которое не поддерживалось в некоторых популярных СУБД реляционной ориентации, например в xBase-системах, также введено в

новые версии этих систем (например, в Visual FoxPro; причем возможно наличие, как изолированных файлов БД, так и объединенных понятием БД. При этом некоторые виды обработки возможны только для файлов, включенных в БД, например, задание ограничений целостности по связи.).

15. Преобладающий язык запросов - SQL. Это тоже говорит о повышении уровня реляционности.

16. Для структурированных БД - увеличение разнообразия типов поддерживаемых полей.

17. Развитие объектно-реляционных и объектно-ориентированных СУБД.

18. Развитие средств обеспечения безопасности данных.

19. Развитие средств упрощения эксплуатации баз данных. Управление базой данных в значительной степени автоматизировано.

20. Повышение качества CASE-средств. Широкое использование CASE-средств в практике проектирования и перепроектирования БнД.

21. Широкое использование графических интерфейсов; визуализация процессов проектирования и управления системой.

1.4.2. Общая характеристика проблемы выбора СУБД

Выбор СУБД является чрезвычайно сложной проблемой, как в научно-методическом, так и в практическом плане. Осуществить строго формализованный выбор СУБД практически невозможно.

Факторы, влияющие на выбор СУБД, можно разделить на ряд групп. Прежде всего, можно выделить факторы, характеризующие саму СУБД и программные средства ее окружения. Другая группа факторов связана с инфраструктурой, сложившейся вокруг каждого из программных продуктов. Третья группа связана с особенностями предполагаемого использования СУБД.

Системы управления базами данных являются сложными языково-программными продуктами. Для их характеристики используется большое число разнообразных качественных и количественных параметров. Причем число и конкретный перечень параметров будут зависеть от класса СУБД (так, например, перечень параметров для характеристики клиент-серверных и настольных систем будет различным). Каждый из классификационных признаков БнД, рассмотренных выше, оказывает влияние на выбор СУБД.

Возможны разные подходы к выбору СУБД. Часто используется упрощенный подход, заключающийся в том, что определяется класс требуемой СУБД, а затем выбирается наиболее популярная на данный момент система данного класса.

Для более обоснованного выбора СУБД необходимо иметь список СУБД-претендентов с описанием их параметров. Желательно, чтобы начальный список был достаточно полным.

Характеристики СУБД рассматриваются с разной степенью детализации в зависимости от стоящих перед проектировщиком задач.

Необходимо определить набор свойств, которым обязательно должна соответствовать выбираемая СУБД (например, обеспечивать должный уровень безопасности, функционировать в определенной операционной среде, поддерживать заданные информационные технологии и др.). Чем более широким и конкретным будет этот список требований, тем уже будет круг выделенных СУБД-претендентов для дальнейшего выбора. Но, с другой стороны, априорное определение такого набора требований может быть не совсем обоснованным.

Обычно сначала осуществляется предварительный отбор СУБД-претендентов по качественным параметрам, а только потом - по количественным. Когда говорят о количественных показателях, чаще всего речь идет о времени выполнения тех или иных операций обработки данных, хотя количественные характеристики этим не ограничиваются.

Среди количественных характеристик можно указать и требуемый объем памяти, как для программного обеспечения, так и для собственно БД, и ограничения, налагаемые СУБД на максимальный объем БД, и максимальное число одновременно поддерживаемых транзакций и др. Часто используются и сложные показатели, такие, как стоимость хранения единицы информации, стоимость выполнения приведенной операции или стоимость «владения» информацией.

Количественные показатели можно определять как расчетным, так и экспериментальным путем.

При определении временных характеристик СУБД чаще всего речь идет о тестах на быстродействие. *Формальное* тестирование заключается в том, что на некотором заданном наборе данных выполняются некоторые операции или наборы операций. Эти операции задаются в терминах СУБД (индексация, чтение с произвольным поиском, удаление и т.п.). Чаще всего такое тестирование проводится либо самими разработчиками СУБД, либо, что лучше, независимыми тестовыми лабораториями. Опубликованные результаты такого тестирования могут помочь в выборе СУБД большому числу потенциальных пользователей.

Функциональное тестирование [37] состоит в том, что исследуются характеристики СУБД при решении определенной прикладной задачи, для реализации которой и выполняется выбор СУБД. При этом требуется реализовать заданные функции. Этот подход дает более точные результаты, однако он очень трудоемкий, что осложняет его практическое использование.

1.4.3. Факторы влияния на выбор СУБД

Отметим важные факторы для выбора СУБД помимо уже названных выше.

1. Платформы, на которых функционирует СУБД.
2. Совместимость с другими системами, открытость, масштабируемость.
3. Уровень языковых средств:
   - трудоемкость изучения;

- трудоемкость создания системы;
- гибкость, выразительная сила;
- мощность;
- наличие языков разного уровня в рамках одной системы.

4. Функциональные возможности.

5. Обеспечение безопасности.

6. Обеспечение целостности.

7. Удобство интерфейса. Наличие интеллектуальных модулей подсказок.

8. Требования к техническим средствам, операционной среде.

9. Ограничения, накладываемые СУБД.

10. Возможности создания «отчуждаемых» приложений. И. Степень универсальности.

12. Локализация.

13. Качество документации.

14. Устойчивость работы, степень отлаженности системы.

15. Наличие средств автоматизации проектирования. Трудоемкость проектирования и перепроектирования.

16. Стоимость СУБД.

17. Мода, тенденции развития информационных технологий (ИТ). Примыкающими к этой группе факторов являются:

- фирма-разработчик;
- распространенность СУБД;
- условия поддержки.

Последний фактор, хотя непосредственно и не характеризует СУБД, но существенно влияет на процесс ее внедрения и эксплуатации.

Поясним некоторые из перечисленных выше характеристик.

Безопасность. Существует несколько уровней (классов) безопасности информационных систем. Вопросы информационной безопасности чрезвычайно важны и в настоящее время, в эпоху всевозрастающей информатизации, являются жизненно важными как для государства в целом, так и для каждого пользователя ИТ. Нужно четко представлять опасности, которые таят в себе информационные технологии, способы решения проблем безопасности и другие связанные с этим вопросы. Определение уровня безопасности системы является очень сложным и дорогостоящим процессом[*].

Фирма-разработчик. При выборе фирмы-разработчика обычно рекомендуют отдавать предпочтение устойчивым фирмам, которые себя уже зарекомендовали на рынке. Значительная доля истины в таких рекомендациях

[*] Так, например, сертификация в России сервера баз данных Orac1e8 на аппаратной платформе HP с операционной системой Unix на соответствие высшему на сегодняшний день классу систем безопасности 1в обошлась Oracle в 1 млн. долл. и длилась полтора года. За это время было проведено 12 тыс. штатных и примерно 5 тыс. специализированных испытаний. [Зырянов М. СУБД в законе // Computerword Россия. - № 23 (136), 23 июня 1998 г.].

есть. Но, с одной стороны, рынок динамично развивается, и оценить степень устойчивости любой фирмы достаточно трудно. С другой стороны, если бы все пользователи руководствовались таким подходом при выборе программных средств, то прогресс бы остановился. Тем не менее, нельзя не обращать внимания на то, что и два других перечисленных ниже фактора также зависят от положения фирмы на рынке.

Распространенность СУБД. Учет этого фактора важен не только с точки зрения косвенной оценки самой СУБД, но и в аспекте кадрового обеспечения. При использовании широко распространенных СУБД организация имеет больше возможностей в подборе нужных кадров и не находится в жесткой зависимости от уникального специалиста. С другой стороны, каждый из специалистов имеет больше степеней свободы в своей профессиональной деятельности.

Условия поддержки. Среди наиболее важных критериев следует отметить наличие консультационной службы на фирме-разработчике программного обеспечения, объем предоставляемых ею услуг и условия их предоставления; наличие фирм-партнеров и/или независимых фирм, выполняющих работы по проектированию систем в среде данной СУБД; возможность пройти обучение и т.п.

СУБД могут быть ориентированы на разные категории пользователей, и соответственно требования к ним будут отличаться. Так, если СУБД позиционируется как инструментальное средство для разработчиков прикладных систем, то значимыми факторами будут наличие и качество компилятора, средства отладки, документирования проекта, поддержка групповой работы проектировщиков и другие возможности, существенные при создании отчуждаемых проектов. Некоторые СУБД предоставляются для продажи в двух редакциях (стандартной и профессиональной).

Все факторы должны рассматриваться во взаимосвязи. Так, например, быстродействие может быть обусловлено не только эффективностью выполнения функций, но, например, отсутствием каких-то возможностей (например, контроля целостности). Показатель быстродействия, к примеру, существен при больших объемах данных или сложных вычислениях и может быть практически незначим при реализации простых запросов на небольших объемах данных. На быстродействие, помимо характеристик самой СУБД, оказывают влияние очень много факторов: характеристика технических средств, подбор параметров, грамотное проектирование, учесть и точно определить взаимовлияние которых на конечный результат достаточно сложно.

Важную группу составляют факторы, характеризующие предметную область, для которой будет создаваться информационная система:

- масштаб системы (объем БД, число пользователей);
- характер обработки информации;
- требования ко времени реакции системы, безопасности данных.

На выбор СУБД будут также оказывать влияние квалификация сотрудников и наличие предшествующих наработок.

При наличии существующей автоматизированной системы обработки информации (а это наиболее частая ситуация) необходимо уяснить, чем вызвана необходимость перехода с существующей системы на новую, установить возможность решения возникших проблем с помощью новых программных средств, определить имеющиеся варианты организации перехода (возможность переноса данных и программ в новую среду, возможность совместной эксплуатации старых и новых программных средств в гетерогенной среде и пр.), а также затраты, связанные с этим.

При анализе предметной области следует не просто зафиксировать ее текущее состояние, а определить тенденции ее развития. Рекомендуется приобретать СУБД «на вырост», иначе может случиться, что менять СУБД придется слишком часто.

1.4.4. Выбор СУБД

Естественно, что при выборе СУБД важно не просто оценить ее, но и определить значимость и нужность той или иной ее возможности (характеристики) для каждой конкретной создаваемой информационной системы. Так, например, для небольших сетей при небольшом объеме данных и отсутствии реальной необходимости в их защите, может быть, и нет особой нужды выбирать систему с архитектурой «клиент-сервер»; для ИС с преобладанием простых поисковых запросов нет смысла выбирать систему, обеспечивающую выполнение многомерного анализа данных, и т.д. Каждый инструмент должен использоваться в адекватных условиях.

Для предварительного отбора СУБД-претендентов можно рекомендовать следующую последовательность шагов:

1) составить перечень требований, которым СУБД должна обязательно удовлетворять;

2) из списка СУБД-претендентов удалить системы, не удовлетворяющие этим требованиям;

3) оставшиеся системы оценить по балльной системе;

4) осуществить окончательный выбор системы из числа СУБД, получивших наибольшие баллы интегральных оценок.

Для оценки СУБД можно построить граф (рис. 1.21), в котором в систематизированном виде представлены существенные при выборе СУБД характеристики. Для каждого СУБД-претендента следует каждой висячей вершине приписать балл, отражающий качество выполнения данной функции в данной СУБД. Веса вершин более высоких уровней рассчитываются на их основе путем вычисления средней величины баллов вершин, соединенных с данной вершиной.

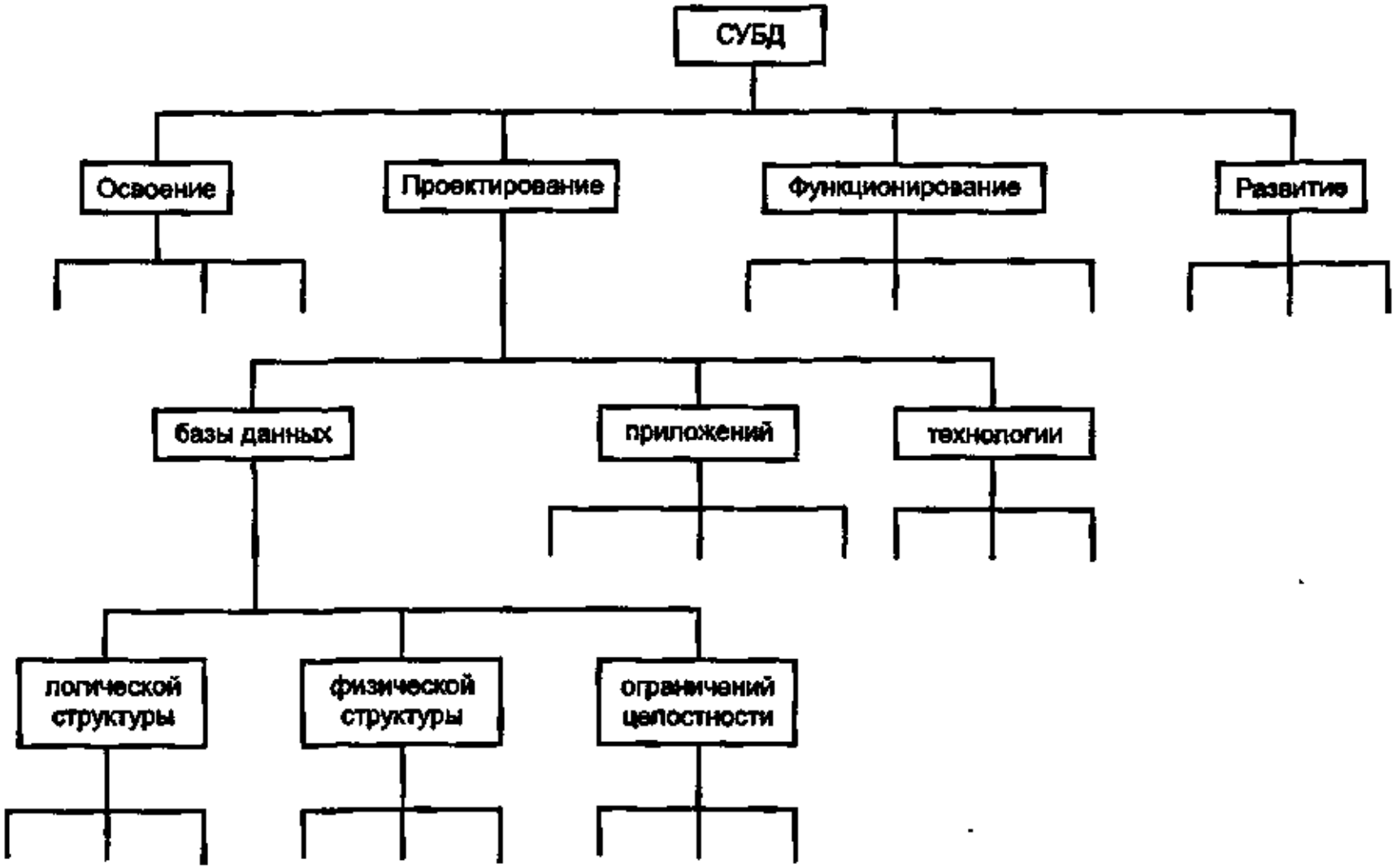


Рис. 1.21. Фрагмент обобщенного графа оценки СУБД

Для учета значимости тех или иных характеристик СУБД любой вершине нужно присвоить второй параметр - коэффициент значимости каждой характеристики для данного приложения. Сумма коэффициентов вершин, принадлежащих одной вышестоящей вершине, равна 1. Для каждой вершины вычисляется взвешенный показатель качества путем умножения балла на коэффициент. Показатель качества вышестоящей вершины определяется как сумма взвешенных показателей подчиненных вершин. Более подробно эта методика выбора СУБД изложена в [19]. Подобный подход достаточно широко используется при выборе не только СУБД. Так, похожая методика была предложена для выбора правовых баз данных [37]. Однако реально использовать ее для выбора СУБД достаточно сложно из-за трудности получения достоверных исходных данных.

Учитывая тенденции развития ИС, наблюдающиеся в настоящее время, можно предположить, что СУБД должна отвечать следующим требованиям:

- обеспечивать работу в гетерогенной сетевой среде, включая возможность эффективной работы в Интернете (для этого СУБД должна поддерживать как общепринятые стандарты сетевого обмена (TCP/ IP, DECnet, IPX/SPX, NetBIOS, SNA и т.д.), так и стандарты межпрограммных интерфейсов (ATMI, XA, ODBC, CORBA));
- легко переноситься с платформы на платформу;
- обеспечивать работу с большими объемами разнотипных данных;
- быть надежной и эффективной.

Выбор СУБД является важным шагом при создании автоматизированных информационных систем (АИС), влияющим на эффективность, как проектирования, так и функционирования системы.

1.5. Уровни моделей и этапы проектирования БД

1.5.1. Уровни моделей

В базе данных отражается информация об определенной предметной области. *Предметной областью* называется часть реального мира, представляющая интерес для данного исследования.

В автоматизированных информационных системах отражение предметной области обеспечивается посредством информационной модели. Мы будем рассматривать далее вопросы проектирования баз данных для СУБД, поддерживающих структурированные модели данных. В зависимости от аспекта рассмотрения (уровня абстракции) различают модели данных нескольких уровней. Число реально выделенных и самостоятельно поддерживаемых уровней моделей будет зависеть от особенностей СУБД *(см.* разд. 1.3.2).

Чаще всего выделяют три уровня моделей: логический, физический и внешний.

Даталогическая (datalogical) модель (ДЛМ) базы данных является моделью логического уровня и представляет собой отображение логических связей между элементами данных безотносительно к среде хранения. Эта модель

строится в терминах информационных единиц, допустимых в той конкретной СУБД, в среде которой проектируется база данных. Этап создания ДЛМ называется *даталогическим проектированием.* Описание логической структуры базы данных на языке СУБД называется *схемой.*

Для привязки даталогической модели к среде хранения используется модель данных физического уровня (для краткости часто называемая *физической моделью).* Эта модель определяет используемые запоминающие устройства и способы физической организации данных в среде хранения. Модель физического уровня также строится с учетом возможностей, предоставляемых СУБД. Описание физической структуры базы данных называется *схемой хранения.* Соответствующий этап проектирования БД называется *физическим проектированием.* СУБД обладают разными возможностями по физической организации данных, в связи, с чем сложность и трудоемкость физического проектирования, набор выполняемых шагов различаются для конкретных систем.

К числу работ, выполняемых на этапе физического проектирования, относятся: выбор типа носителя, способа организации данных, методов доступа; определение размера физического блока, управление размещением данных на внешнем носителе, управление свободной памятью, определение целесообразности сжатия данных и используемых методов сжатия, оценка физической модели данных. К физическому проектированию относятся и проблемы, связанные с буферизацией.

Независимо от того, поддерживаются ли в явном виде отдельно модели логического и физического уровня, с точки зрения методологии все равно можно выделить эти уровни моделей и соответствующие им этапы проектирования баз данных.

В некоторых СУБД, помимо описания общей логической структуры базы данных, имеется возможность описать логическую структуру БД с точки зрения конкретного пользователя. Такая модель называется *внешней,* а ее описание называется *подсхемой.* Если СУБД поддерживает схему, схему хранения и подсхему, то она является СУБД с трехуровневой архитектурой.

Внешняя модель не всегда является точным подмножеством схемы. Некоторые СУБД допускают различия в типах данных, определенных в схеме и подсхеме, и обеспечивают их преобразование, позволяют задавать различный логический порядок следования элементов в схеме и подсхеме, обеспечивают введение в подсхему виртуальных полей и т.д.. Если определена подсхема, то пользователь имеет доступ только к тем данным, которые отражены в соответствующей подсхеме, что является одним из способов защиты информации от несанкционированного доступа.

В подсхемах часто задается не только логическая структура части базы данных с точки зрения конкретного пользователя (приложения), но и допустимые режимы обработки в рамках этой подсхемы, что служит дополнительным механизмом защиты информации от разрушения.

Использование аппарата подсхем облегчает работу пользователя, так как он должен знать структуру не всей базы данных, а только той ее части, которая имеет непосредственное к нему отношение.

В тех случаях, когда СУБД в явном виде не поддерживает подсхемы, перечисленные функции могут выполнять другие компоненты системы. Близким к понятию подсхемы является понятие view (взгляд), которое в настоящее время широко используется в англоязычной литературе по реляционным СУБД.

Выше мы говорили о трех уровнях моделей, которые поддерживаются СУБД. Но для того чтобы спроектировать структуру базы данных, необходима исходная информация о предметной области. Желательно, чтобы эта информация была представлена в формализованном виде. Такое формализованное описание предметной области (ПО) будем называть *инфологической* (infological) моделью предметной области (ИЛМ) или *концептуальной* моделью (КМ)[*]. Информация, требуемая для проектирования БД, мало зависит от особенностей СУБД. Более того, для проектирования ИС с «небанковской» организацией (но использующей структурированное представление данных) обычно требуется та же исходная информация. Поэтому *концептуальная схема* представляет собой описание предметной области, выполненное без жесткой ориентации на используемые в дальнейшем программные и технические средства. Концептуальная схема должна отражать специфику предметной области, а не структуру БД. Иногда в концептуальную схему добавляют информацию, отображающую чисто языковые характеристики, такие, как наличие синонимов, длина реквизитов и др. Это, скорее всего, вызвано следующими основными причинами:

1) нежеланием вводить еще один уровень моделей;

2) трудностью отделения языковых проблем от других, так как анализируемая предметная область обычно представлена в какой-либо знаковой системе и анализу обычно подвергается именно это представление, а не непосредственно сама ПО.

1.5.2. Взаимосвязь этапов проектирования БД

Начальным шагом проектирования ИС является построение инфологической модели предметной области. Предварительная инфологическая модель строится еще на предпроектной стадии и затем уточняется на более поздних стадиях проектирования. Затем на ее основе строится даталогическая модель. Физическая и внешняя модели после этого могут строиться в любой последовательности по отношению друг к другу, в том числе и параллельно.

---

[*] В предыдущих изданиях учебника [17, 18] для этого уровня моделирования использовался термин «инфологическое». Последнее время в литературных источниках этот термин используется редко. В данном издании учебника термины «инфологическое» и «концептуальное» моделирование используются как синонимы.

На рис. 1.22 показана взаимосвязь этапов проектирования БД. Как видно из рисунка, при проектировании БД возможен возврат на предыдущие уровни. При этом возможны два типа возврата: первый обусловлен необходимостью пересмотра результата проектирования (например, для улучшения полученных характеристик, обхода ограничений и т.п.), второй тип вызван потребностью уточнения предыдущей модели (как правило, инфологической) в целях получения дополнительной информации для проектирования или при выявлении противоречий в модели.

На рис. 1.23 и 1.24 изображены укрупненные технологические сети проектирования для этапов даталогического и физического проектирования*. Как видно из этих рисунков, результат предыдущего этапа проектирования используется на входе следующего этапа.

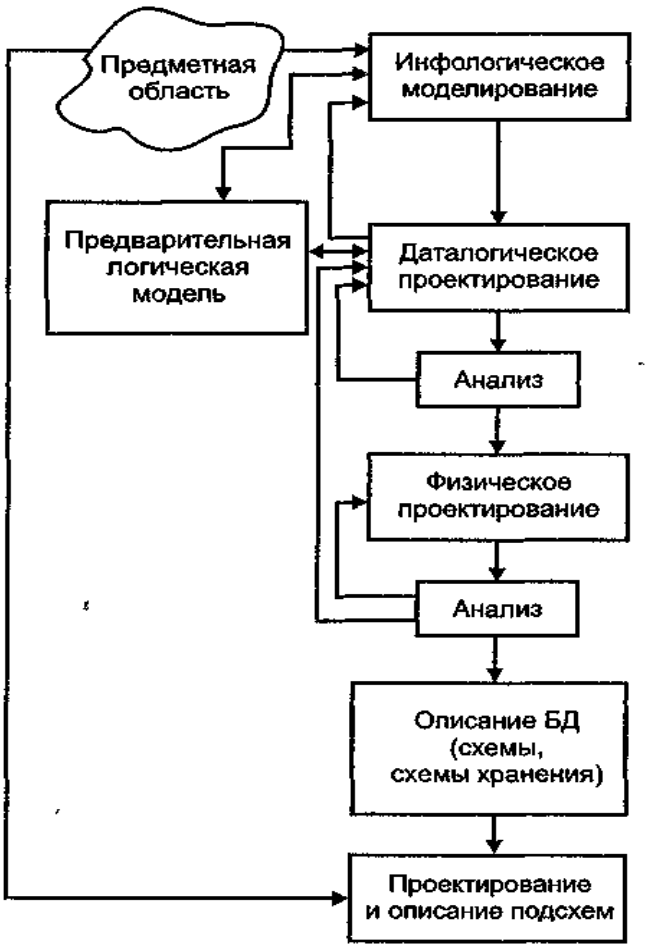


Рис. 1.22. Взаимосвязь этапов проектирования

* Следует обратить внимание на различие терминологии, используемой как в литературных источниках, так и в конкретных CASE-системах. Так, во многих CASE-системах ER-модель предметной области называется концептуальной схемой, а представление логической структуры целевой базы данных - физической моделью.

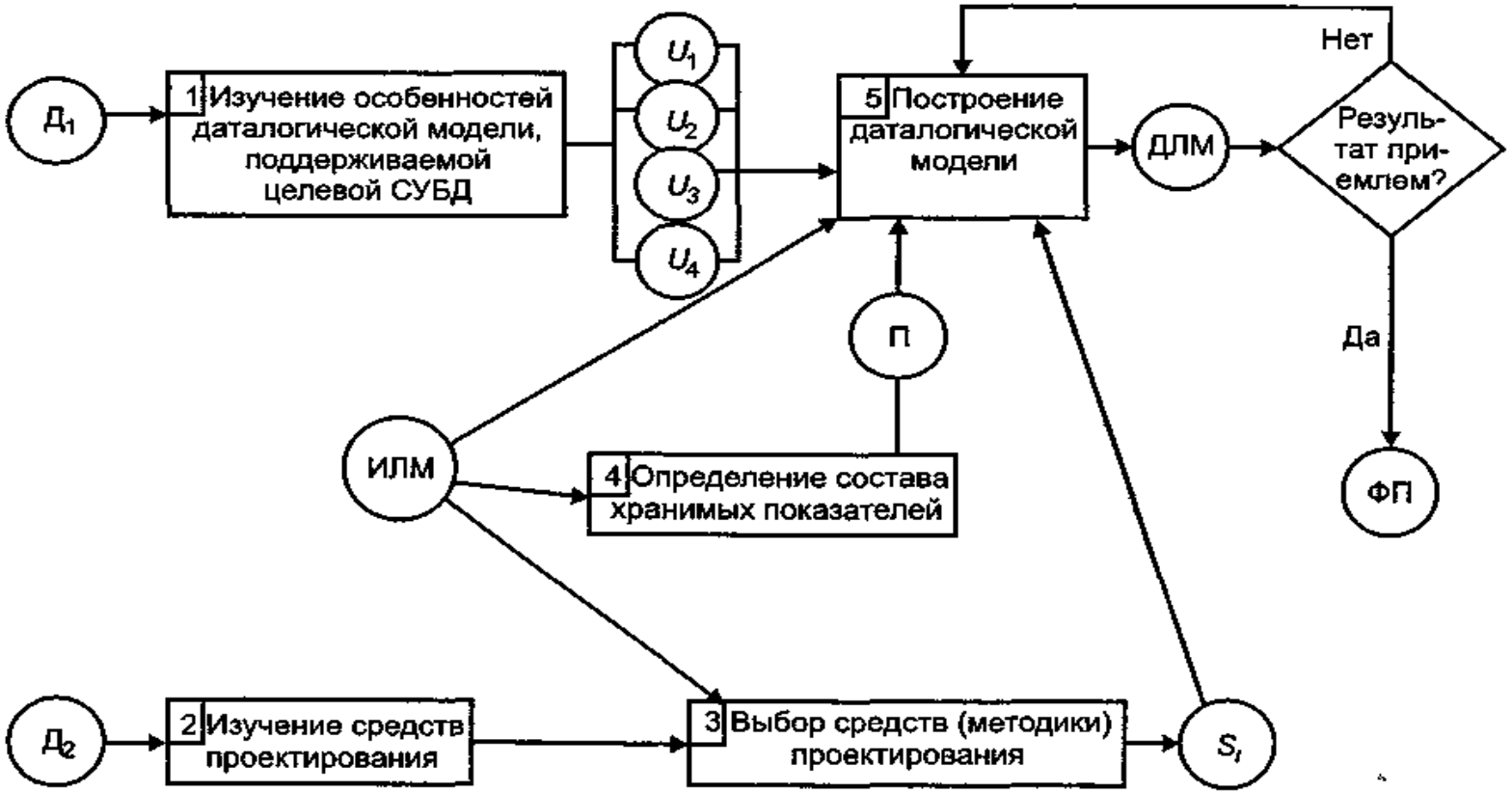


Рис. 1.23. Технологическая сеть проектирования для этапа даталогического проектирования:
$Д_1$ - документация по СУБД; $Д_2$ - документация по средствам проектирования; U_1 - набор допустимых даталогических конструкций; U_2 - операторы ЯМД; U_3 - ограничения, налагаемые СУБД на ДЛМ;
U_4 - возможности физической организации данных; П - перечень хранимых показателей;
S_i- выбранное средство проектирования; ФП - физическое проектирование

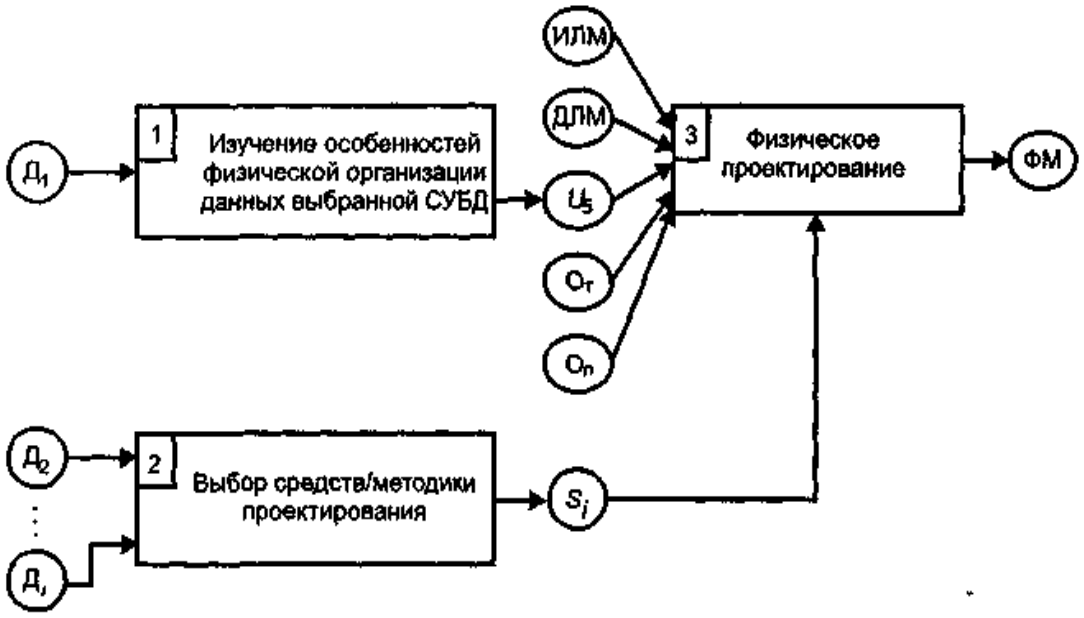


Рис. 1.24. Технологическая сеть проектирования для этапа физического проектирования: U_5 - возможности физической организации данных; $O_т$ - ограничения на используемые технические средства; O_n - ограничения со стороны пользователей (процессов); ФМ - физическая модель

1.5.3. Факторы влияния на проектирование БД

Как было отмечено выше, на стадии инфологического моделирования должна быть собрана и представлена в надлежащем виде вся информация, необходимая и достаточная для дальнейшего проектирования БД. Для того чтобы было понятно, какая информация должна фиксироваться при описании предметной области, перечислим основные факторы, оказывающие влияние на проектирование структуры БД:

1. Специфика предметной области:
   - особенности отображаемых объектов, характер связи между объектами предметной области;
   - «размер» системы (объем хранимых данных).
2. Особенности требуемой обработки информации:
   - характеристика запросов (критерий поиска, частота запроса; состав реквизитов, выдаваемых в ответ, упорядоченность ответа, частота совместного использования реквизитов и т.п.);
   - требования к защите информации;
   - ограничения по времени реакции системы на каждый из запросов, что, в свою очередь, определяется несколькими факторами, такими, как режим выполнения запроса (интерактивный, пакетный, в реальном масштабе времени), статус запроса и др.
3. Характеристика пользователей системы:

- важность (статус), приоритеты;
- число пользователей;
- распределение функций между пользователями, степень пересечения информационных потребностей пользователей;
- приоритеты пользователей в оценке значимости факторов, влияющих на проектирование БД;
- технология обработки данных;
- возможность/необходимость работы в распределенной среде, в том числе необходимость поддерживать связь с мобильными компьютерами;
- доступные технологии обработки данных.

4. Состояние существующей системы обработки информации:

- наличие автоматизированной системы обработки информации;
- объем имеющихся наработок;
- наличие технических и программных средств, их состояние;
- соотношение объемов существующей и новой частей проектируемой системы;
- затраты для перевода имеющейся системы на новую основу.

5. Возможности, предоставляемые используемыми (выбранными для реализации проекта) техническими и программными средствами:

- поддерживаемые структуры данных;
- ограничения, накладываемые программным обеспечением;
- ограничения по объему памяти;
- быстродействие технических средств;
- производительность программного обеспечения;
- особенности языков манипулирования данными.

6. Трудоемкость проектирования.

7. Финансовые возможности.

8. Квалификация кадров:

- разработчиков;
- пользователей.

9. Используемые методики проектирования:

- наличие средств автоматизации проектирования;
- используемый алгоритм проектирования.

10. Субъективные факторы:

- мода;
- привычки и предпочтения.

Более подробно влияние некоторых из перечисленных выше факторов будет рассмотрено далее, по мере изложения вопросов проектирования БД.

На это следует обратить внимание

- Банк данных — это сложная человеко-машинная система, удовлетворяющая определенным принципам.
- Наблюдается большое разнообразие как отдельных компонентов БнД, так и целостных решений на их основе. Необходимо выбрать решения, подходящие в каждой конкретной ситуации.
- СУБД включают средства, ориентированные на разные категории пользователей — начиная с конечного пользователя и кончая администратором БнД.
- Следует опасаться как ложного упрощенчества при создании БнД, так и необоснованного страха перед сложностью встающих проблем.
- Проектирование БнД включает в себя ряд взаимосвязанных этапов. Важно понять значимость каждого из этапов проектирования. Особое внимание следует обратить на этапы моделирования предметной области и проектирования структуры БД, так как эти этапы оказывают наибольшее влияние на качество проекта в целом, а стоимость исправления ошибок, допущенных на этих этапах, особенно велика.

Контрольные вопросы

1. Дайте определение банка данных.
2. Назовите основные преимущества банков данных.
3. Назовите основные недостатки банков данных.
4. Каковы предпосылки создания БнД?
5. Какие требования предъявляются к банкам данных?
6. Какие компоненты включаются в состав банка данных?
7. Что называется системой управления базой данных?
8. Что называется базой данных?
9. Дайте классификацию языковых средств СУБД.
10. Какие поколения языковых средств вы знаете? Дайте краткую характеристику языковым средствам каждого из поколений.
11. Назовите принципы, по которым построены языки 4-го поколения.
12. Перечислите компоненты языка 4-го поколения.
13. Приведите примеры процедурных и непроцедурных языков. В чем основные отличия между языками этих классов?
14. Какие поколения СУБД вы знаете? Дайте краткую характеристику СУБД каждого из поколений.
15. Назовите основные отличительные особенности банков данных.
16. Какие технические средства необходимы для реализации банка данных?

17. Какие типы ЭВМ чаще всего используются для реализации банков данных?

18. Перечислите основные признаки классификации банков данных.

19. В чем разница между системами со структурированными и неструктурированными базами данных?

20. Охарактеризуйте основные классы СУБД.

21. Какие факторы влияют на выбор СУБД?

22. СУБД, каких классов являются в настоящее время наиболее распространенными?

23. Какие основные тенденции развития СУБД наблюдаются в настоящее время?

24. Сравните системы типа OLTP и OLAP.

25. Перечислите основные отличия корпоративных и настольных СУБД.

26. Сравните локальные, интегрированные и распределенные БД.

27. Перечислите этапы проектирования баз данных.

28. Что называется схемой, подсхемой и внешней схемой?

29. Какую роль выполняет подсхема? Какие преимущества дает ее использование?

30. Что называется словарем данных, репозиторием?

31. Охарактеризуйте взаимодействие компонентов БнД при работе с системой.

32. Что называется инфологической моделью?

33. Является ли инфологическое моделирование этапом, присущим только проектированию баз данных?

34. Какая информация является исходной для построения концептуальной модели?

35. Кто должен создавать концептуальную модель и почему?

36. Какие требования предъявляются к инфологической модели?

37. Что называется даталогической моделью?

38. Какая информация является исходной для построения даталогической модели?

39. Какие вопросы решаются на стадии даталогического моделирования?

40. Изобразите технологическую сеть проектирования для стадии даталогического моделирования.

41. Что называется физической моделью?

42. Какая информация является исходной для построения физической модели?

43. Какие вопросы решаются на стадии физического моделирования?

44. Какие факторы влияют на проектирование БД?

45. Перечислите основные категории пользователей банков данных.

46. Кого называют конечными пользователями?

47. Кого называют администраторами банка данных?

48. Перечислите основные функции администратора банка данных.

49. В каком порядке должны выполняться этапы проектирования БнД?

50. Какая информация является исходной при выполнении инфологического проектирования?

51. Какие факторы влияют на проектные решения при создании баз данных?

52. Какие критерии используются при оценке спроектированной БД?

Глава 2 КОНЦЕПТУАЛЬНОЕ ПРОЕКТИРОВАНИЕ

2.1. Общие сведения о моделировании предметной области

Концептуальное проектирование является центральной частью, ядром всего процесса проектирования баз данных. Подходы к концептуальному проектированию, излагаемые в разных литературных источниках и реализованные в разнообразных CASE-системах, отличаются друг от друга. Многие из подходов имеют существенные недостатки, что не позволяет рекомендовать именно их как основные. В связи с этим опишем некоторую базовую модель, с которой проводятся сравнения других систем.

Так как в настоящее время CASE-систем достаточно много, то неизвестно, с какой именно из систем придется проектировщику столкнуться на практике. Поэтому приведем некоторые критерии, по которым следует сравнивать CASE-системы, и обобщенные рекомендации по построению ER-моделей в зависимости от доступных изобразительных средств и алгоритмов проектирования логической структуры базы данных. В качестве примеров рассмотрим процесс концептуального моделирования в среде Design/IDEF и ERWin.

2.1.1. Уточнение понятия концептуальной модели

В базе данных отображается какая-то часть реального мира. Естественно, что полнота ее описания будет зависеть от целей создаваемой информационной системы. Как указано выше, часть реального мира, представляющая интерес для данного исследования, называется *предметной областью.* Для того чтобы база данных адекватно отражала предметную область, проектировщик должен хорошо представлять себе все нюансы, присущие ей, и уметь отобразить их в базе данных.

Предметная область должна быть предварительно описана. Для этого в принципе может использоваться и естественный язык, но его применение имеет много недостатков, основные из которых - громоздкость описания и неоднозначность его трактовки. Поэтому обычно для этих целей используют искусственные формализованные (чаще всего - графические) языковые средства.

Формализованное описание предметной области будем называть ее *концептуальной моделью.* Предметные области могут быть различными, и для их моделирования могут потребоваться специфические средства, соответствующие особенностям этих областей. Мы будем ориентироваться в основном на экономико-организационные системы, хотя описываемые далее подходы к проектированию являются более универсальными и могут быть использованы и в других предметных областях.

Моделирование предметных областей выполняется с разными целями, например для реинжиниринга бизнесс-процессов, для прогнозирования раз-

вития предметной области, при проектировании баз данных и программного обеспечения и т.п. Используемые средства и методы моделирования при этом будут различаться. Мы остановимся здесь только на средствах и методах моделирования, которые находят наибольшее использование при проектировании баз данных.

Как было отмечено в главе 1, существует большое разнообразие видов БД. Подходы к проектированию баз данных разных классов будут существенно различаться. Поскольку в настоящее время основную часть баз данных представляют структурированные базы данных, то основное внимание будет уделено проектированию именно таких систем.

Изучение предметной области складывается из непосредственного наблюдения протекающих в ней процессов, изучения документов, циркулирующих в системе, а также интервьюирования участников этих процессов (рис. 2.1). Так как описание инфологической модели выполняется на специализированном языке, то необходимо владение этим языком. Следует обратить внимание на то, что возможности языка описания ИМЛ оказывают влияние на методику построения модели с использованием данных языковых средств. Построение концептуальной модели может выполняться и вручную, и с использованием автоматизированных средств проектирования. Средства автоматизации проектирования отличаются как нотациями используемых языковых средств, так и алгоритмами преобразования концептуальной модели в модели базы данных. Это, в свою очередь, скажется на методике построения модели в их среде.

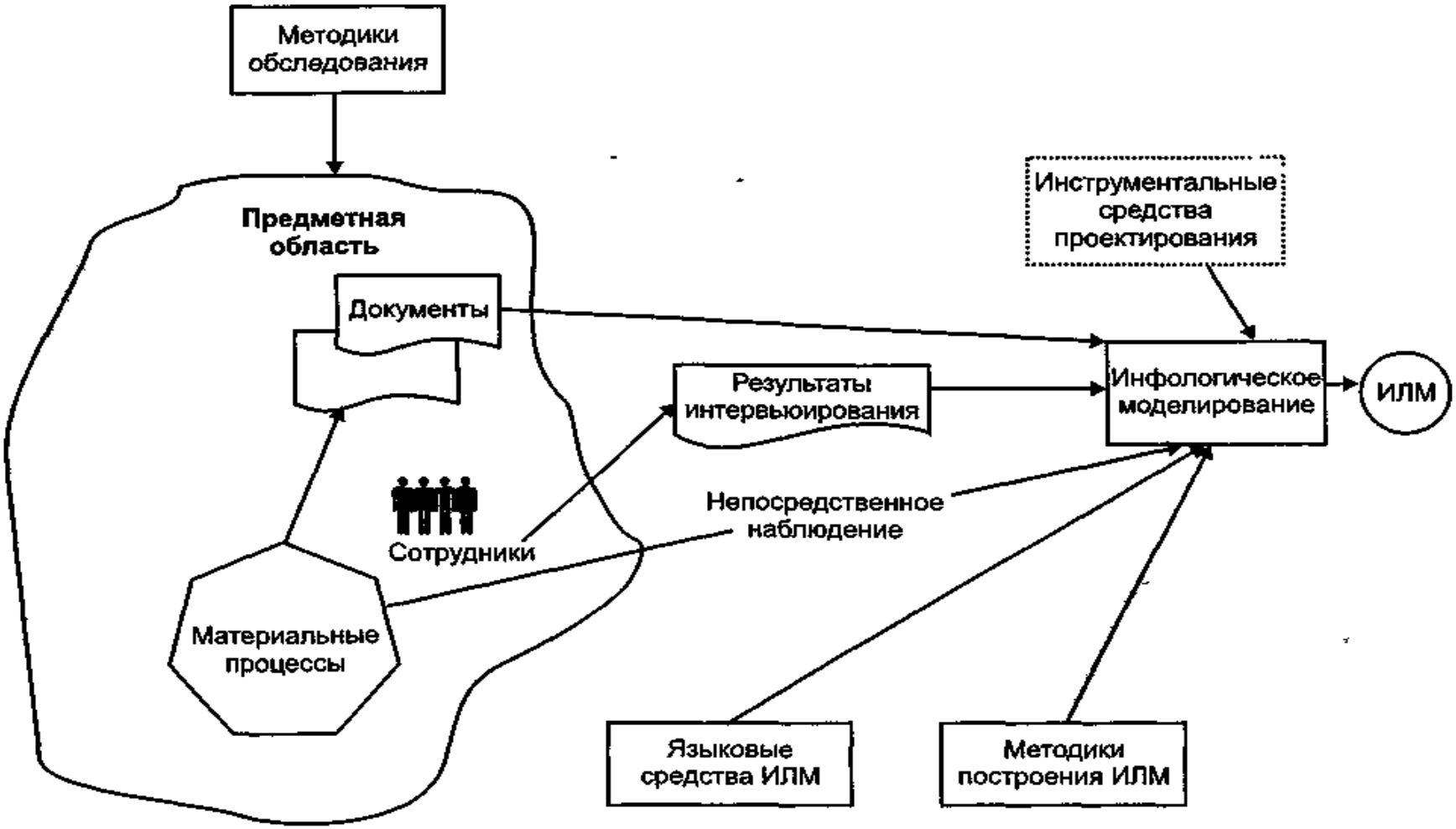


Рис. 2.1. Стадия инфологического моделирования - исходная и результатная информация

2.1.2. Основные компоненты концептуальной модели

Основными компонентами концептуальной модели ПО являются (рис. 2.2):

• описание объектов ПО и связей между ними;

• описание информационных потребностей пользователей;

• описание существующей информационной системы (документы, документооборот, при наличии автоматизированной информационной системы - ее описание);

• описание алгоритмических зависимостей показателей;

• описание ограничений целостности;

• описание функциональной структуры системы, для которой создается АИС;

• требования к ИС и существующие ограничения;

• лингвистические отношения.

Далее более подробно остановимся на первом из перечисленных компонентов, так как именно он оказывает наибольшее влияние на проектирование структуры базы данных. Чаще всего описание объектов ПО и связей между ними представляется в виде так называемых ER-моделей (или ER-диаграмм)*.

2.1.3. Требования, предъявляемые к концептуальной модели

К концептуальной модели предъявляются следующие требования:

- адекватное отображение предметной области (язык для представления модели должен обладать достаточными выразительными возможностями для отображения явлений, имеющих место в предметной области, а сама модель должна содержать всю необходимую и достаточную информацию для дальнейшего проектирования системы);
- непротиворечивость (модель отражает взгляды и потребности всех пользователей системы, а также обычно является результатом работы многих специалистов, поэтому целостное описание ПО должно быть проверено на непротиворечивость);
- однозначная трактовка модели всеми ее пользователями (обеспечивается формализованностью языка и четким его пониманием всеми участниками процесса создания ИС);
- легкость восприятия разными категориями пользователей (обеспечивается выбором соответствующего языка моделирования);

* ER (Entity-Relationship) - сущность-связь.

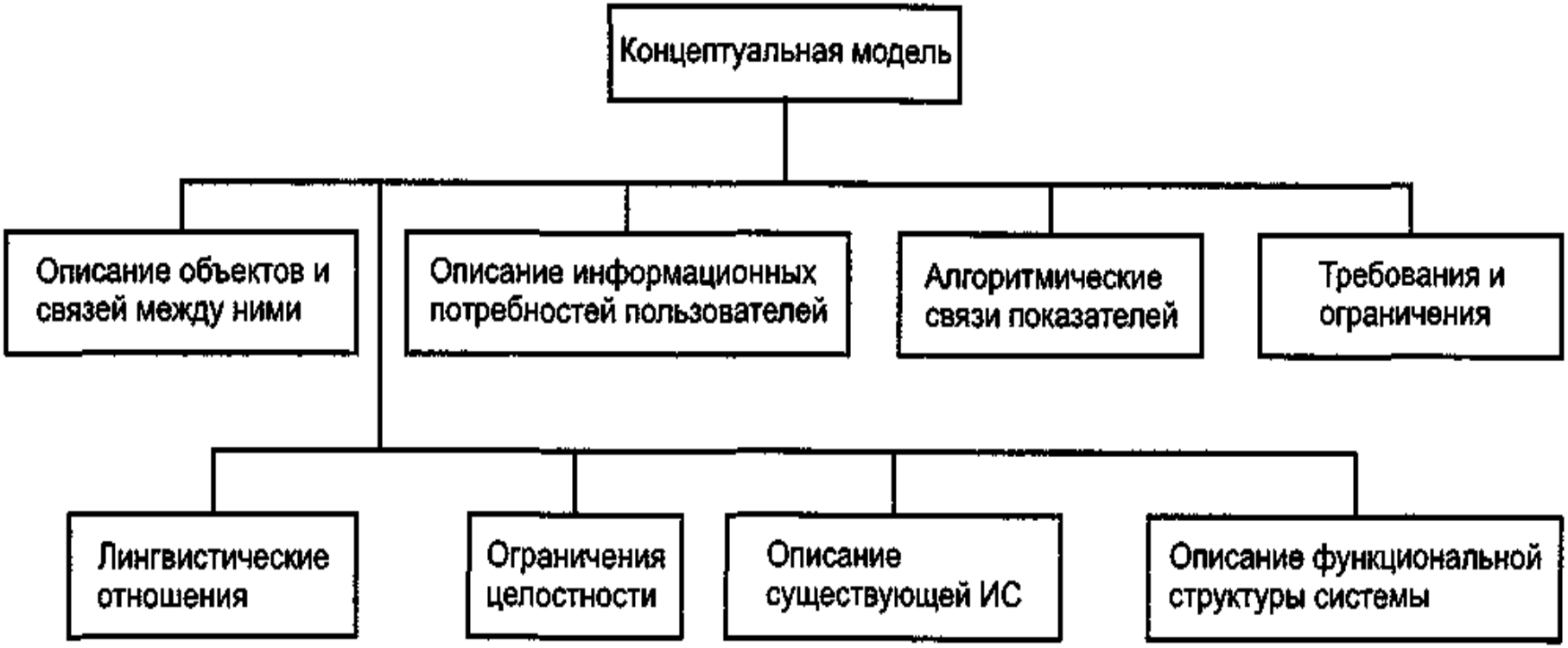


Рис. 2.2. Компоненты концептуальной модели

- конечность модели (несмотря на то, что реальный мир, отображаемый в КМ, является по своей природе бесконечным, инфологическая модель является конечной, что обеспечивается четким ограничением предметной области);
- легкость модификации (в концептуальную модель по разным причинам часто приходится вводить новые объекты или модифицировать существующие; КМ должна в связи с этим обладать свойством легкой расширяемости, обеспечивающим ввод новых данных без изменения раннее определенных. То же самое можно сказать и об удалении и корректировке данных);
- возможность композиции и декомпозиции модели.

Желательно, чтобы язык спецификации концептуальной модели был одинаковым как при ручном, так и при автоматизированном проектировании информационных систем. Последнее предъявляет к языку дополнительные требования, а именно, он должен:

- быть вычисляемым, т.е. восприниматься и обрабатываться ЭВМ;
- использовать «дружелюбные» пользователю интерфейсы, в частности графические;
- быть независимым от оборудования и других ресурсов, которые подвержены частым изменениям;
- использовать средства тестирования КМ, а также иметь аппарат для указания того, что спецификация завершена и по ней может быть выполнена генерация структур баз данных.

При автоматизированном проектировании все изменения, внесенные в КМ, должны быть автоматически отражены в связанных с модифицируемым элементом компонентах банка данных.

Желательно, чтобы КМ строили специалисты, работающие в предметной области, для которой создается АИС, а не проектировщики систем машинной обработки данных. Если в силу определенных причин это невозможно, то необходимо, чтобы первые могли хотя бы проверить сделанное другим специалистом описание, чтобы убедиться в том, что специфика предметной области воспринята и отображена правильно.

Концептуальная модель является средством коммуникации разнообразных коллективов, как конечных пользователей, так и разработчиков. Информация из КМ корреспондирует со словарной системой и другими компонентами банка данных.

2.1.4. Преимущества использования ER-моделирования

ER-модель представляет собой графическое описание предметной области в терминах «объект – свойство – связь». ER-модель является одним из элементов концептуальной модели. Использование ER-моделирования (особенно в сочетании с автоматизированными средствами проектирования - CASE-средствами) дает много преимуществ:

- предписывая определенную методологию моделирования, делает анализ предметной области более целенаправленным и конкретным;
- является удобным средством документирования проекта;
- позволяет вести проектирование АИС без привязки к конкретной целевой СУБД и осуществлять выбор последней в любой момент времени (чем ближе к концу проектирования это будет сделано, тем точнее может быть выбор).

При использовании ER-моделирования в составе CASE-средств появляются дополнительные преимущества:

- снижаются требования к знанию деталей языков описания данных (DDL - Data Definition Language) и диалектов SQL конкретных СУБД;
- при смене используемой СУБД не нужно проводить проектирование заново; следует только осуществить шаг по переводу ER-модели в целевую (если выбранная вами целевая СУБД поддерживается данным CASE-средством, то такой переход вообще будет выполнен автоматически);
- наличие в CASE-средстве возможности «обратного проектирования» (т.е. получения ER-диаграммы по имеющимся описаниям данных) позволяет использовать существовавшие ранее наработки для «реверс-инжиниринга» системы;
- указание связи объектов в ER-модели и соответствующая миграция ключа при преобразовании этой модели в целевую не только позволяют задавать контроль целостности связи при ведении БД, но и автоматически обеспечивают согласованное описание схемы (внешний ключ мигрирует в связанное отношение; при этом имя, тип и длина соответствующего атрибута повторяются в зависимой сущности);
- сокращается время проектирования системы;
- появляется возможность автоматизированного тестирования проекта на всех этапах проектирования;
- повышается качество документирования проекта;
- мощные современные CASE-средства позволяют вести коллективную разработку проекта.

2.2. Описание базовой ER-модели

Существует большое число нотаций (изобразительных средств) и методик построения ER-моделей. Рассмотрим методику построения ER-модели, предлагаемую автором данного учебника. Будем называть эту методику ER-моделирования *базовой.* Некоторые другие методики изложены в разд. 2.3, а также в [1-3, 7, 15, 20, 21, 25] или в документации по CASE-средствам.

2.2.1. Понятия «объект» и «класс объектов»

В предметной области имеется множество разнообразных объектов. Объект – понятие широкое. Его трудно точно определить. Обычно под объектом понимают некую сущность (реальную или абстрактную), о которой собирается какая-то информация. Объекты группируются в классы. *Классом объектов* называют совокупность объектов, обладающих одинаковым набором свойств. Например, для объектов класса СТУДЕНТ таким набором свойств являются: «Год_рождения», «Пол» и др.

Объекты могут быть реальными, как названный выше объект СТУДЕНТ, и абстрактными, как, например, ПРЕДМЕТЫ, которые изучают студенты.

ER-модель строится на уровне классов объектов, а не отдельных экземпляров объектов.

Каждому классу объектов в ER-модели присваивается уникальное имя. Именем класса объекта является грамматический оборот существительного (существительное, у которого могут быть прилагательные и предлоги). Если имя состоит из нескольких слов, то желательно, чтобы первым стояло существительное. Существительное должно употребляться в единственном, а не во множественном числе (например, ДИСЦИПЛИНА_ИЗУЧАЕМАЯ). Если в предметной области традиционно используются разные имена для обозначения какого-либо класса объектов (т.е. имеет место синонимия), то все они должны быть зафиксированы при описании системы, и затем одно из них выбирается за основное, и только оно должно в дальнейшем использоваться в ER-модели. Помимо имени класса объектов в ER-мо-дели может использоваться его короткое кодовое обозначение; для дальнейшего перехода к даталогической модели еще может указываться имя, которое будет использоваться при описании структуры базы данных.

При построении ER-модели желательно дать словесную интерпретацию каждой сущности, особенно если возможно неоднозначное толкование понятия.

Вместо термина «объект» часто используется термин «сущность». В дальнейшем будем рассматривать эти термины как синонимы.

При отражении в информационной системе каждый объект (имеется в виду уже экземпляр объекта, а не весь класс) представляется своим именем, которое называет конкретный объект и отличает один объект от другого. Чтобы выполнять свою роль, имя должно быть уникальным, но иногда в реальной жизни это бывает не так (явление синонимии). Поэтому в концептуальной модели должны быть каким-то образом обозначены случаи, когда естественное имя объекта является неуникальным. Уникальное имя объекта будем называть *идентификатором* (ПО). Каждый объект должен иметь, по крайней мере один идентификатор.

Каждый объект обладает определенным набором *свойств* - характеристик, описывающих состояние каждой сущности. Набор (перечень) свойств для всех объектов данного класса будет одинаковым, но конкретные

значения этих характеристик и особенно сочетание этих значений будут отличаться от объекта к объекту, что, собственно, и отличает один экземпляр объекта от другого.

Может случиться, что в данной предметной области свойства объекта не представляют для нас интереса. В связи с этим в ER-модели возможно наличие объектов, не имеющих свойств и представленных только своими идентификаторами.

2.2.2. Разновидности объектов

Различают несколько разновидностей объектов. Прежде всего это простые и сложные объекты. Объект называется *простым,* если он рассматривается в данном исследовании как неделимый.

Сложный объект представляет собой объединение других объектов, простых или сложных, также отображаемых в информационной системе. Понятия «простой» и «сложный» объект являются относительными. В одном случае объект может считаться простым, а в другом - этот же объект может рассматриваться как сложный. Так, например, объект АУДИТОРИЯ, если АИС строится только для управления учебным процессом, будет рассматриваться как простой. Если же АИС будет включать подсистемы для служб энергетика, материально-технического снабжения и др., то АУДИТОРИЯ будет рассматриваться как составной объект.

Выделяют несколько разновидностей сложных объектов: составные, обобщенные и агрегированные объекты.

Составной объект соответствует отображению отношения «целое-часть». Примерами составных объектов являются УЗЕЛ-ДЕТАЛИ, КЛАСС-УЧЕНИКИ и т.п.

Обобщенный объект отражает наличие связи «род-вид» между объектами предметной области. Например, объекты СТУДЕНТ, ШКОЛЬНИК, АСПИРАНТ, УЧАЩИЙСЯ_ТЕХНИКУМА образуют обобщенный объект УЧАЩИЙСЯ. Объекты, составляющие обобщенный объект, называются его *категориями.*

Как родовой объект, так и видовые объекты могут обладать определенным набором свойств. Причем наблюдается так называемое наследование свойств, т.е. видовой объект обладает всеми теми свойствами, которыми обладает родовой объект, плюс свойствами, присущими только объектам этого вида.

Определение родовидовых связей означает классификацию объектов предметной области по тем или иным признакам. Естественно, что классификация может быть многоуровневой.

Агрегированные объекты соответствуют обычно какому-либо процессу, в который оказываются вовлеченными другие объекты. Например, агрегированный объект ПОСТАВКА объединяет в себе объекты ПОСТАВЩИК, ПОТРЕБИТЕЛЬ, а также саму поставляемую ПРОДУКЦИЮ. Своеобразным объектом является ДАТА_ПОСТАВКИ. Агрегированный объект может, так

же как и простой объект, иметь характеризующие его свойства. В рассматриваемом примере таким свойством может быть «Размер_поставки». Агрегированные объекты обычно выражаются отглагольными существительными.

2.2.3. Изображение простого объекта

Для графического обозначения простого объекта будем использовать прямоугольник, ограниченный сплошной линией. Название класса объекта пишется над ним. Внутри прямоугольника записывается название атрибута, именующего объект (рис. 2.3). Если у объекта имеется несколько имен, то для каждого из них выделяется отдельный сектор в этом прямоугольнике.

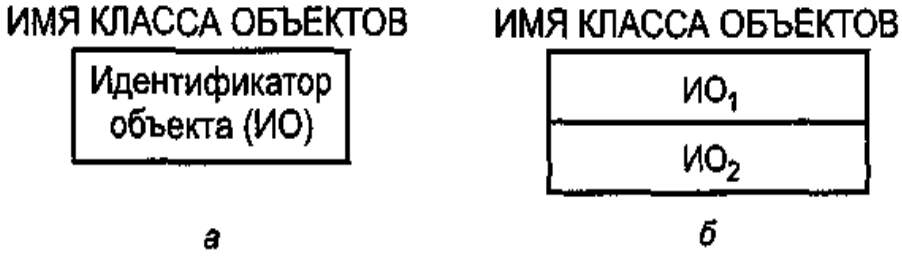


Рис. 2.3. Изображение объекта: *а* - с одним идентификатором; *б* — с несколькими

Если какое-либо имя объекта не является уникальным, будем использовать букву «н» рядом с таким именем (рис. 2.4).

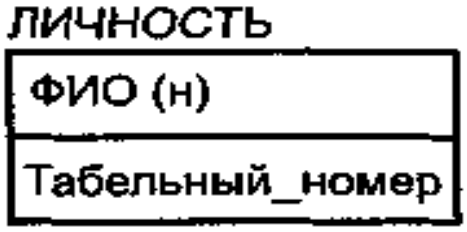


Рис. 2.4. Пример изображения объекта с несколькими идентифицирующими атрибутами

Встречаются случаи, когда идентификация одних объектов зависит от идентификации других. Например, часто для участков цехов предприятия используется не сквозная нумерация, а в пределах каждого цеха, т.е. участок имеет составной идентификатор «Номер_це-ха*Номер_участка». Назовем подобные объекты (в нашем примере это объект УЧАСТОК) *зависимыми по идентификации сущностями.* Для отображения таких ситуаций (рис. 2.5, 2.6) будем перечеркивать линию, соединяющую соответствующие объекты, около конца, прилегающего к зависимому объекту (ЗО).

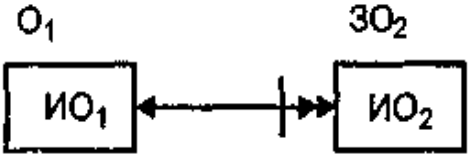


Рис. 2.5. Изображение зависимой по идентификации сущности (условные обозначения)

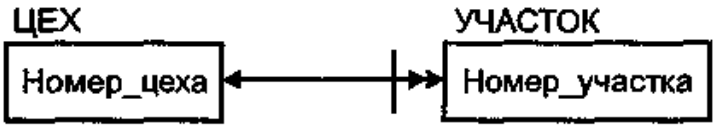


Рис. 2.6. Пример изображения зависимой по идентификации сущности

Если не использовать специального обозначения для указания зависимости по идентификации, то (для нашего примера) для объекта УЧАСТОК следует указать составной идентификатор «Номер_цеха*Номер_участка», при этом его надо указать в одном секторе прямоугольника, а не выделять несколько секторов, как в случае наличия у объекта нескольких имен.

2.2.4. Описание свойств объекта. Разновидности свойств

Как указывалось выше, класс объектов представляет собой совокупность объектов, обладающих одинаковым набором свойств. При описании предметной области нужно изобразить набор свойств, фиксируемых для объектов каждого из представленных в модели классов. Для обозначения свойств будем использовать прямоугольник, изображенный пунктирной линией.

Связь между объектом и характеризующим его свойством изображается в виде линии, соединяющей их обозначения. Характер связи между объектом и его свойством может быть различный. Объект может обладать только одним значением какого-то свойства в каждый момент времени. Например, каждый человек может иметь только одну «Дату_рождения» или «Стаж_работы». Назовем такие свойства *единичными.* Для других свойств возможно существование одновременно нескольких значений у одного и того же объекта (например, свойство «Иностранный_язык» у объекта СОТРУДНИК, если СОТРУДНИК может владеть несколькими иностранными языками). Такое свойство будем называть *множественным.* При изображении связи между объектом и его свойствами для единичных свойств будем использовать одинарную стрелку, а для множественных свойств - двойную стрелку на конце линии, соединяющей объект с данным свойством (рис. 2.7, 2.8).

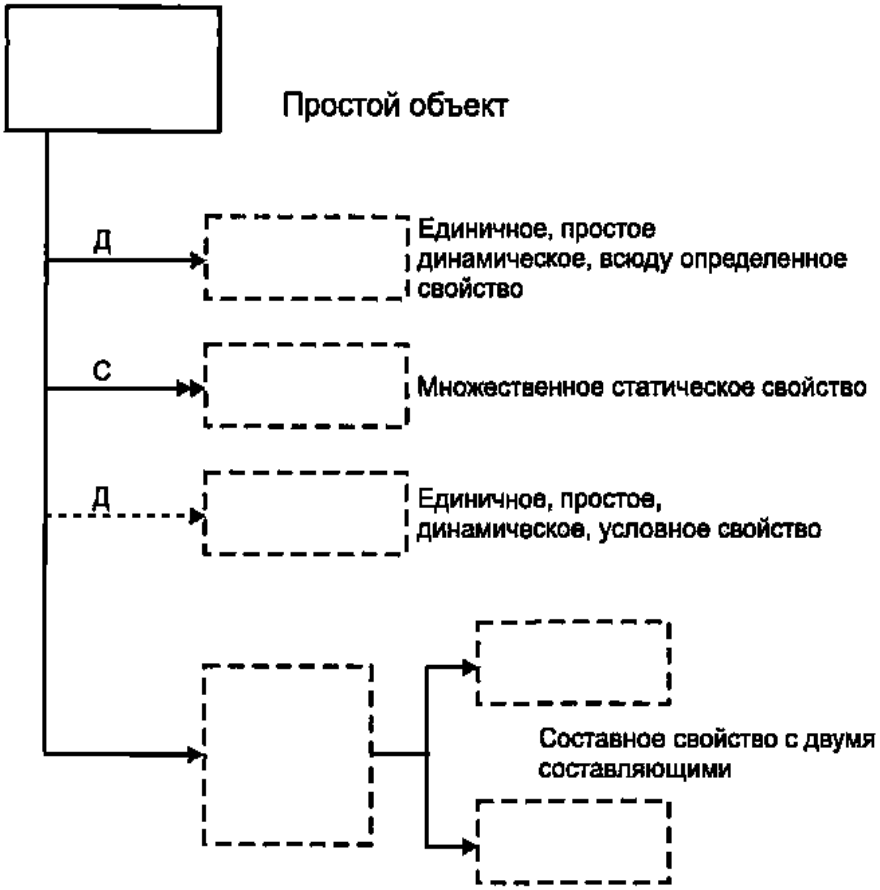


Рис. 2.7. Изображение объекта и его свойств (условные обозначения)

Значения некоторых свойств не может измениться с течением времени. Назовем такие свойства *статическими,* а те свойства, значения которых могут изменяться со временем, будем называть *динамическими.* Для обозначения динамических свойств будем использовать букву «Д», а статических - «С» над соответствующей линией. Так, упомянутое выше свойство «Дата_рождения» будет являться статическим, а «Стаж» - динамическим.

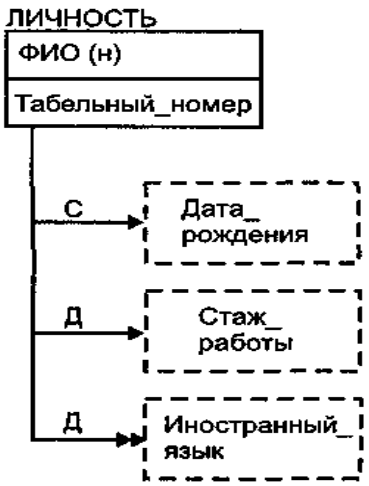


Рис. 2.8. Пример изображения единичных и множественных динамических и статических свойств

Другой характеристикой связи между объектом и его свойством является признак того, присутствует ли это свойство у всех объектов данного класса либо оно может отсутствовать у некоторых объектов. Например, для отдельных служащих может иметь место свойство «Ученая_степень», а другие объекты этого класса могут не обладать указанным свойством. Назовем свойства, присутствующие не у всех объектов данного класса, *условными.* При изображении связи условного свойства с объектом будем использовать пунктирную линию, а если свойство определено для всех экземпляров объектов данного класса - сплошную (*см.* рис. 2.7, 2.9).

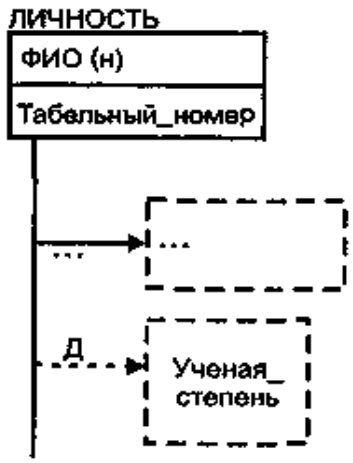


Рис. 2.9. Пример изображения условного свойства

Правильность отображения предметной области в ER-модель будет зависеть от того, какие ситуации возможны в данной предметной области, а какие - нет. Так, если в вузе сотрудник может занимать несколько должностей одновременно, например быть одновременно ректором и заведующим кафедрой, то фрагмент ER-модели будет выглядеть так, как изображено на рис. 2.10, *а,* а если внутривузовское совместительство не разрешено - то так, как изображено на рис. 2.10, б.

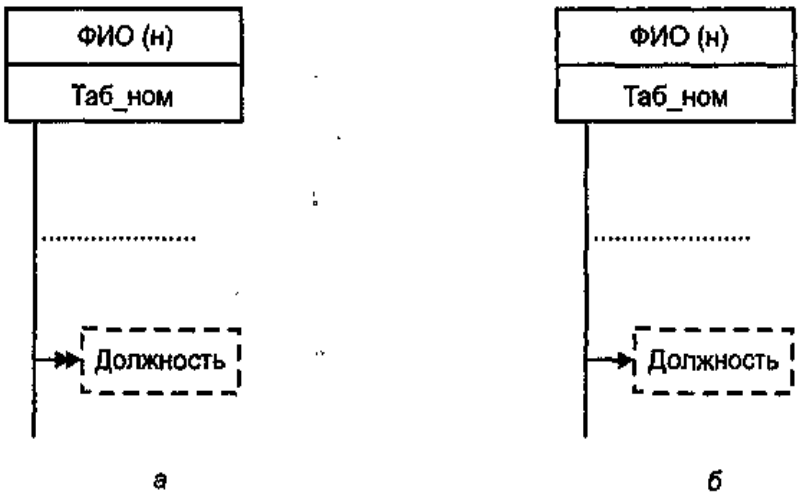


Рис. 2.10. Варианты изображения предметной области:
а - разрешено совместительство; *б* - не разрешено

Иногда в ER-модели бывает полезно ввести понятие *составного свойства.* Примером такого свойства могут быть «Адрес», состоящий из «Города», «Улицы», «Дома» и «Квартиры». Будем использовать для обозначения составного свойства пунктирный квадрат, из которого исходят линии, соединяющие его с обозначениями составляющих его элементов (*см.* рис. 2.7, 2.11).

При проектировании БД определяются тип и длина полей. Для того чтобы иметь возможность правильно выбрать эти характеристики, необходимо иметь соответствующую информацию о типе представления атрибута в «немашинной» системе и требования/пожелания пользователей об их отображении в автоматизированной системе, может быть, даже с предпочтениями. Например, предположим, что желательно было бы хранить в БД изображение. Если целевая СУБД не позволяет это сделать, то возможны следующие варианты:

- поле, соответствующее данному атрибуту, не вводить;
- связать БД с системой, которая может хранить рисунок;
- заменить рисунок описанием.

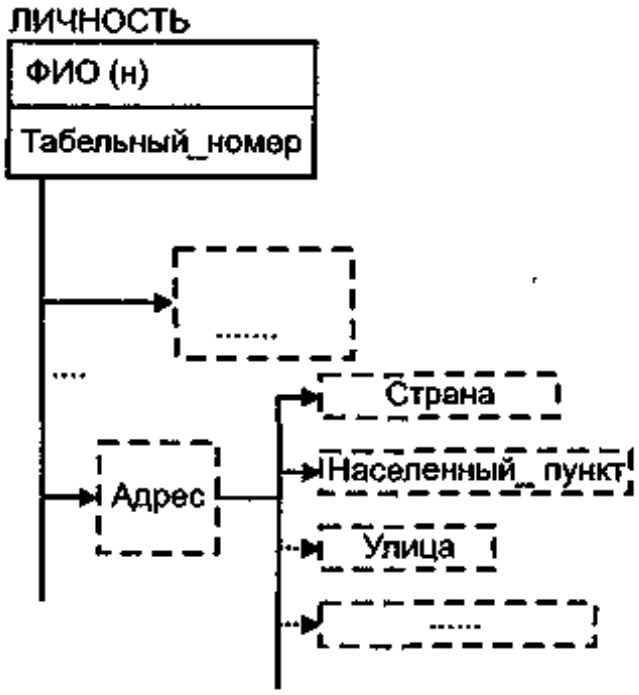


Рис. 2.11. Пример изображения составного свойства

Например, в «Листке по учету кадров» хранится фотография. Если есть возможность ее сканирования и связи соответствующего файла с записями БД, то сделать это, если нет - то все, что соответствует фотографии, не хранить в ИС.

Для всех реквизитов символьного типа должна быть указана их максимальная длина (а лучше - не только максимальная, но и минимальная, и средневзвешенная).

Чтобы не загромождать ER-модель, подобные характеристики (табл. 2.1) рекомендуется отображать в репозитории (в каталоге реквизитов).

Понятия «объект» и «свойство» являются относительными. Что в каждой из моделей ПО следует считать самостоятельным объектом, а что - свойством другого объекта, будет зависеть от аспекта рассмотрения данной предметной области. Например, пусть строится АИС для управления конкретным учебным заведением. Для СОТРУДНИКОВ и УЧАЩИХСЯ указывается, какое учебное заведение они закончили. Больше никакой информации об учебных заведениях не хранится; никакой специальной обработки по этому признаку не проводится. В этом случае не стоит выделять отдельный объект «УЧЕБНОЕ_ЗАВЕДЕНИЕ», а следует считать его свойством соответствующего объекта. Если же в предметной области отражается дополнительная информация об учебных заведениях, например их адрес, тип и т.п., то УЧЕБНОЕ_ЗАВЕДЕНИЕ следует рассматривать как самостоятельный объект.

Таблица 2.1

Наименование характеристики	Что отражает?	Возможные значения	Пример*
Назначение	Что определяет?	Идентификатор (обозначает, называет объект) Качественная характеристика Количественная характеристика Дата/время совершения события Изображение	ФИО Пол Вес Дата рождения Фотография
Изменяемость значения свойства	Может ли меняться в течение жизненного цикла объекта?	Динамическое Статическое	Образование Дата рождения
Обязательность	Может ли отсутствовать данное свойство у объекта?	Обязательное Необязательное	Дата рождения Ученая степень
Элементарность	Возможность разбиения на составляющие элементы	Простое Составное	Пол Адрес
Множественность	Возможность наличия нескольких значений для данного свойства у одного объекта	Единичное Множественное	Дата рождения Номера телефонов

Продолжение таблицы 2.1

Форма отображения в знаковой системе		Символьная Числовая Дата Время Изображение Логическое	Адрес Вес Дата рождения Время прихода на работу Фотография сотрудника Военнообязанный (да/нет)
Способ получения		Датчики/счетчики Из внешней среды Производная информация	Кардиограмма Рекомендации с прежней работы Стаж работы
* Примеры приведены для класса объектов ЛИЧНОСТЬ.			

В экономических организационных системах большая часть информации отражается в символьном виде. При этом различают реквизиты-признаки, определяющие качественные характеристики объектов, и реквизиты-основания, отражающие количественные характеристики объектов. Некоторые характеристики могут быть получены различными способами, в том числе путем вычисления из других хранящихся в информационной системе показателей. Такие показатели называются *производными.* При проектировании БД необходима информация о возможных способах получения каждого реквизита для решения вопроса о том, какая информация должна в явном виде храниться в БД, а какая может быть получена путем преобразования имеющейся информации.

2.2.5. Алгоритмические зависимости

В инфологической модели должны быть отражены алгоритмические зависимости между показателями. Обычно для этих целей используются графы взаимосвязи показателей, отражающие, какие показатели служат исходными для вычисления других (рис. 2.12). Расчетные формулы и алгоритмы вычислений также в том или ином виде должны быть представлены в ИЛМ.

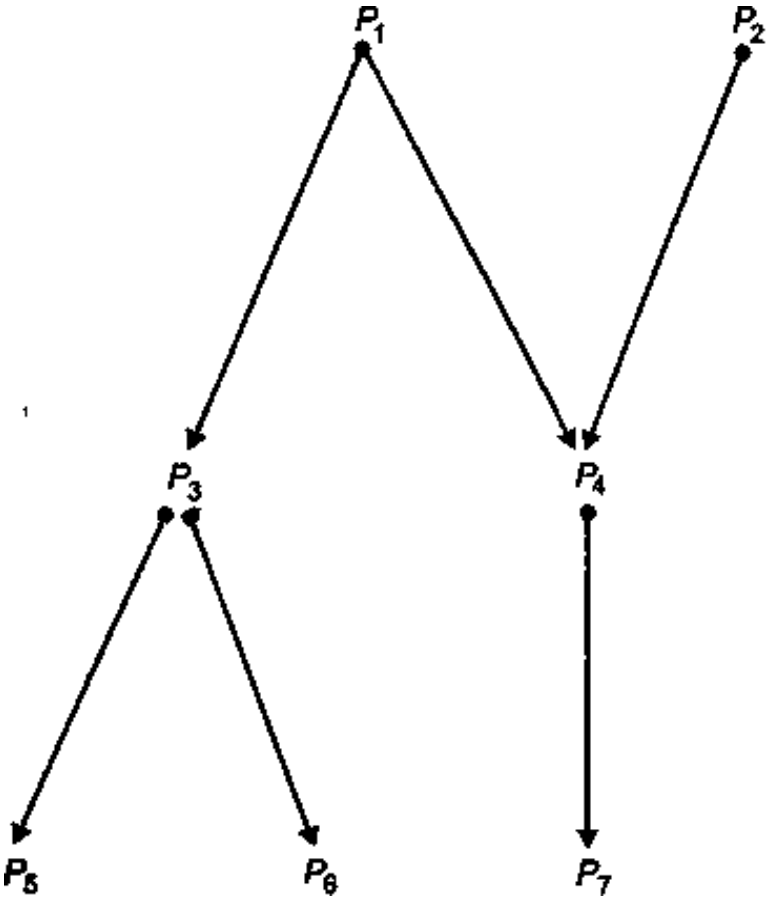


Рис. 2.12. Граф взаимосвязи показателей

Следует обратить внимание на то, что производными могут быть не только количественные, но и качественные характеристики. Например, для объекта СТУДЕНТ имеется характеристика «Отличник». Значение этой характеристики определяется по следующему правилу: отличником считается студент, защитивший все курсовые работы/проекты на «отлично», сдавший все экзамены на «отлично», а также выполнивший в заданный срок все остальные предусмотренные учебным планом контрольные мероприятия.

Информация об алгоритмических зависимостях не является элементом ER-модели, но является необходимым компонентом описания предметной области. Эта информация используется для определения состава хранимых в базе данных показателей, определения ограничений целостности, реализации бизнес-правил.

2.2.6. Интегральные характеристики класса объектов

Как отмечалось выше, в ER-модели отображаются не отдельные экземпляры объектов, а классы объектов. Когда в ER-модели изображено обозначение объекта, то ясно, что речь идет о классе объектов, обладающих описанными свойствами. Поэтому в эти модели в большинстве случаев можно в явном виде не вводить еще и обозначение для класса объектов. Явное изображение класса объектов необходимо только в том случае, если в предметной

области для данного класса объектов фиксируются не только характеристики, относящиеся к отдельным объектам этого класса, но и какие-то интегральные характеристики, относящиеся ко всему классу в целом. Например, если для класса объектов СОТРУДНИК_ПРЕДПРИЯТИЯ фиксируется не только возраст каждого сотрудника, но и средний возраст всех сотрудников, то в ER-модели необходимо отразить не только объект СОТРУДНИК, но и класс объектов СОТРУДНИКИ. Для отображения класса объектов лучше использовать какое-то специальное обозначение; в нашем случае это прямоугольник, очерченный двойной линией (рис. 2.13).

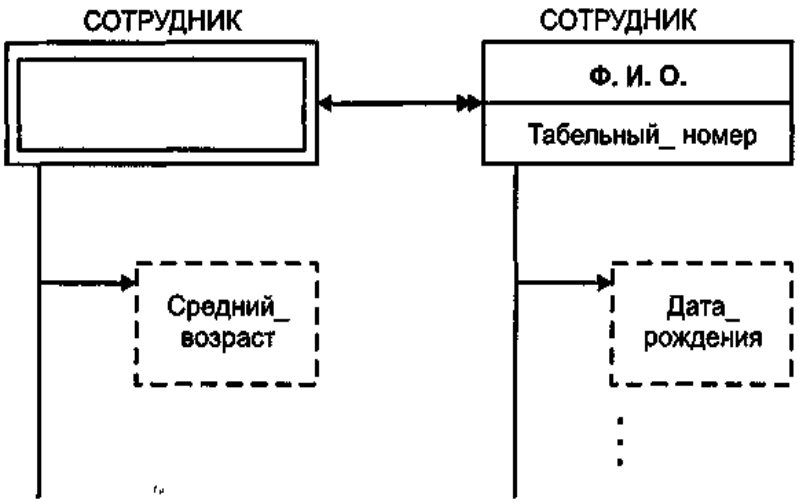


Рис. 2.13. Отображение интегральных характеристик класса объектов

Для характеристики каждого класса объектов полезно указать число объектов в классе, а также динамику его изменения. Чаще всего указывается число объектов в классе на начало периода, а также число прибывающих и убывающих объектов за фиксированный период. Для того чтобы не перегружать графическое изображение ER- модели, будем эту информацию отображать в отдельных таблицах. Например, если предметной областью является вуз, имеющий стабильный ежегодный набор студентов, то количественная характеристика класса объектов СТУДЕНТ может иметь вид, представленный в табл. 2.2. Если число объектов в классе или динамика изменения не являются постоянными, то для соответствующих показателей желательно фиксировать максимальное, минимальное и среднее число объектов в классе.

Количественные характеристики классов объектов используются не только для определения объема памяти, занимаемого БД, но и для обоснованного принятия решений по организации данных. Знание динамики изменения объектов в классе дает информацию, необходимую для принятия решений по организации данных и технологии их обработки. Так, число объектов в классе СТУДЕНТ (*см.* табл. 2.2) является постоянным (5000), но ежегодно 1/5 часть студентов оканчивает институт и столько же новых студентов поступает. Это означает, что ежегодно добавляется 20% общего объема дан-

ных о студентах, и такой же объем данных должен быть перенесен в архивные файлы. Без наличия информации о динамике изменения класса объектов необходимость принятия соответствующих решений была бы просто не видна.

Таблица 2.2

Код класса объектов	Наименование класса	Определение	Код родительского класса	Число объектов в классе	Динамика изменения	
					+	-
1	Кадры	Все лица, которые либо работают, либо обучаются в институте	-	15 000		
2	Учащиеся		1	11200		
3	Студенты		2	5000	1000	1000
4	Слушатели подготовительных курсов		2	6000	6000	6000
5	Аспиранты		2	200	80	80
6	Сотрудники		1	500		
7	Преподаватели		6	300	10	20
8	Вспомогательный персонал		6	200	60	80

2.2.7. Связи между объектами

Кроме связи между объектом и его свойствами в ER-модели фиксируются связи между объектами разных (а иногда - одного и того же) классов. *Связь* (Relationship) - это ассоциация между сущностями, при которой каждый экземпляр одной сущности ассоциирован с произвольным (в том числе нулевым) количеством экземпляров другой сущности. Обычно рассматриваются *бинарные связи,* т.е. связи между двумя классами объектов. Связи являются двунаправленными. Связи могут устанавливаться и между сущностями одного класса. Например, связь «Быть_руководителем» устанавливается и между разными экземплярами объектов одного класса СОТРУДНИК.

Отображение типа связи. Выделяют различные виды связи (рис. 2.14): «один к одному» (1:1), «один ко многим» (1:М) и «многие ко многим» (М:М).

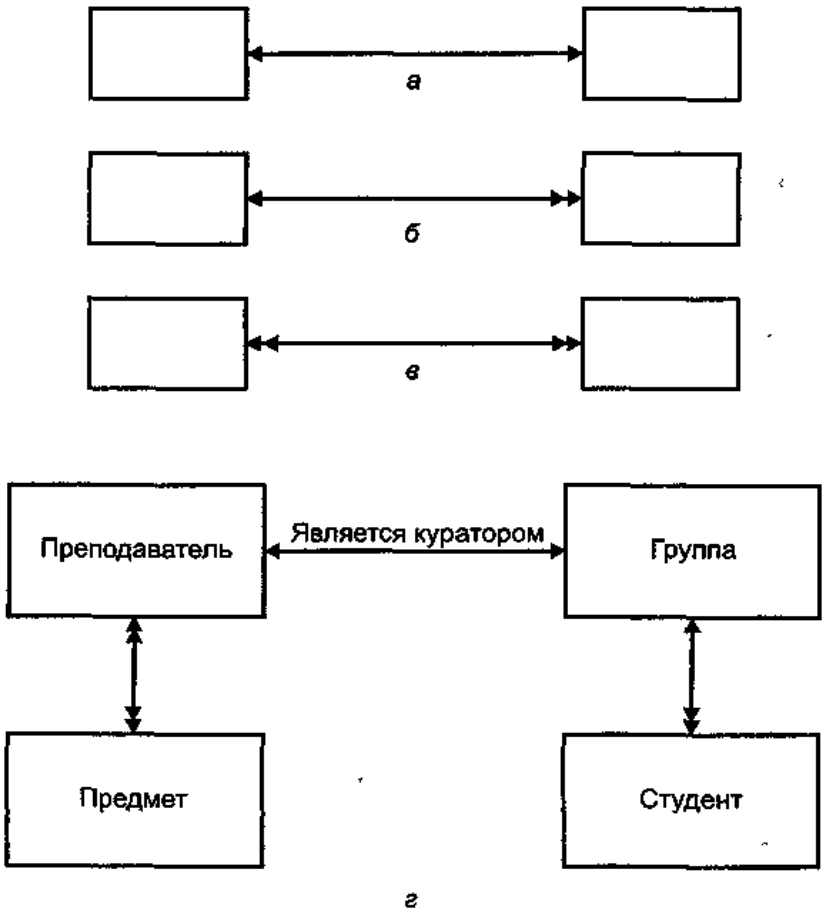


Рис. 2.14. Виды связи между объектами: *а* - отношение 1:1; *б* - 1:М; *в* - М:М; *г* - пример использования связи

Если связь множественная, то желательно еще указать и мощность связи: число объектов М (это может быть минимальное, максимальное и среднее число объектов в связи).

Возможны случаи, когда между парой объектов объявлено несколько связей (рис. 2.15). В этом случае связи следует именовать (указание роли)*.



Рис. 2.15. Пример объявления двух связей между парой объектов

* Будем придерживаться следующего правила: если роль связи очевидна, то ее имя можно не указывать на схеме. Но в любом случае указание этого имени (или, наоборот, его отсутствие) не является ошибкой. Указание имени связи делает модель более информативной, легкой для понимания, но не оказывает влияния на алгоритм проектирования БД.

Отображение класса принадлежности. Кроме степени связи в ER-модели для характеристики связи между разными сущностями нужно указывать так называемый *класс принадлежности* (обязательность связи), который показывает, может ли отсутствовать связь объекта данного класса с каким-либо объектом другого класса. Класс принадлежности сущности может быть либо *обязательным,* либо *необязательным.*

Аналогично с условным свойством необязательное вхождение объекта в связь будем обозначать пунктирной линией **с** той стороны, объекты которой могут не входить в соответствующую связь. Таким образом, линия, соединяющая объекты, может быть полностью сплошной, полностью пунктирной либо наполовину - сплошной, наполовину - пунктирной (рис. 2.16).

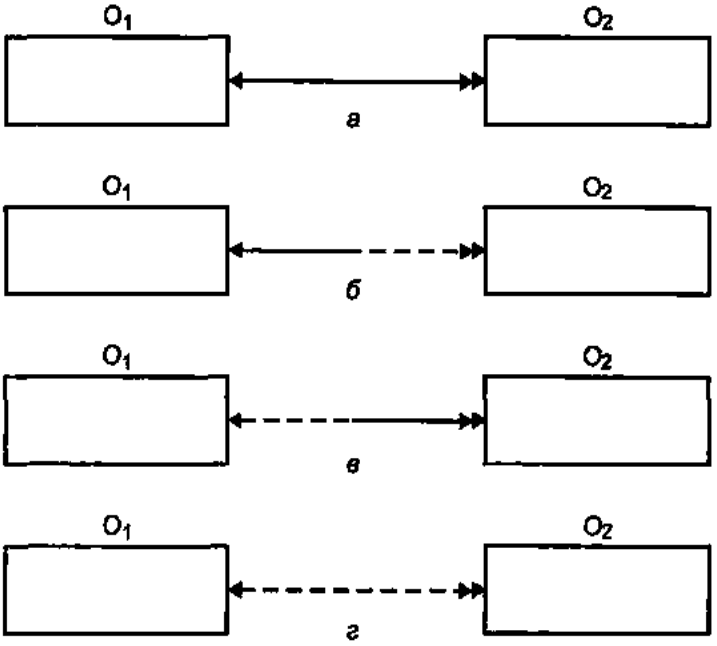


Рис. 2.16. Варианты классов членства: обязательный класс членства с обеих сторон (а); необязательный класс членства с правой стороны *(б),* с левой стороны *(в),* с обеих сторон *(г)*

Поясним сказанное на конкретных примерах. Для графической иллюстрации используем так называемые диаграммы ER-экземпляров (вместо диаграмм ER-типа, которые мы использовали до сих пор).

Мы хотим отобразить в инфологической модели связь между двумя классами объектов: ЛИЧНОСТЬ и ЯЗЫК_ИНОСТРАННЫЙ.

Предположим, что предметной областью является завод, некоторые сотрудники которого знают какой-либо иностранный язык, но ни один из них не владеет более чем одним языком. Есть некоторые сотрудники, которые не владеют ни одним языком. Естественно, что имеется много языков, которыми не владеет ни один из сотрудников, а также что некоторые из сотрудников владеют одним и тем же иностранным языком. В этом случае диаграмма ER-экземпляров будет иметь вид, изображенный на рис. 2.17, *а,* диаграмма ER-типов - как на рис.2.17, *б.* Тип связи - 1:M (на диаграмме это отображено со

стороны объекта ЛИЧНОСТЬ двойной стрелкой, а со стороны объекта ЯЗЫК_ИНОСТРАННЫЙ - одинарной стрелкой на линии, изображающей связь между рассматриваемыми сущностями), класс принадлежности - необязательный с обоих концов (линия - полностью пунктирная).

Предположим далее, что предметной областью является институт, а объект ЛИЧНОСТЬ отображает абитуриентов, поступающих в этот институт. Каждый из абитуриентов обязательно должен владеть каким-либо иностранным языком, но никто не владеет более чем одним языком; предположение, что имеется много языков, которыми не владеет ни один из сотрудников, остается актуальным и в этой ситуации. В этом случае диаграмма ER-экземпляров будет иметь вид, изображенный на рис. 2.18, *а,* диаграмма ER-типов - как на рис. 2.18, *б.*

Как в первом, так и во втором случае между сущностями наблюдается отношение 1:М. Разница в рассматриваемых ситуациях заключается в том, что в первом случае класс принадлежности является необязательным для обеих сущностей, а во втором - для сущности ЛИЧНОСТЬ класс принадлежности обязательный. На диаграмме это будет отображено пунктирной линией, прилегающей к объекту ЯЗЫК_ИНОСТРАННЫЙ, и сплошной линией, прилегающей к объекту ЛИЧНОСТЬ.

Пусть предметная область будет та же, что и в предыдущем случае, но имеют место ситуации, когда некоторые абитуриенты знают несколько иностранных языков. Связь между объектами при этом будет иметь тип М:М. Для такой предметной области диаграмма ER-экземпляров будет иметь вид, изображенный на рис. 2.19, *а,* диаграмма ER-типов - как на рис. 2.19, *б.*

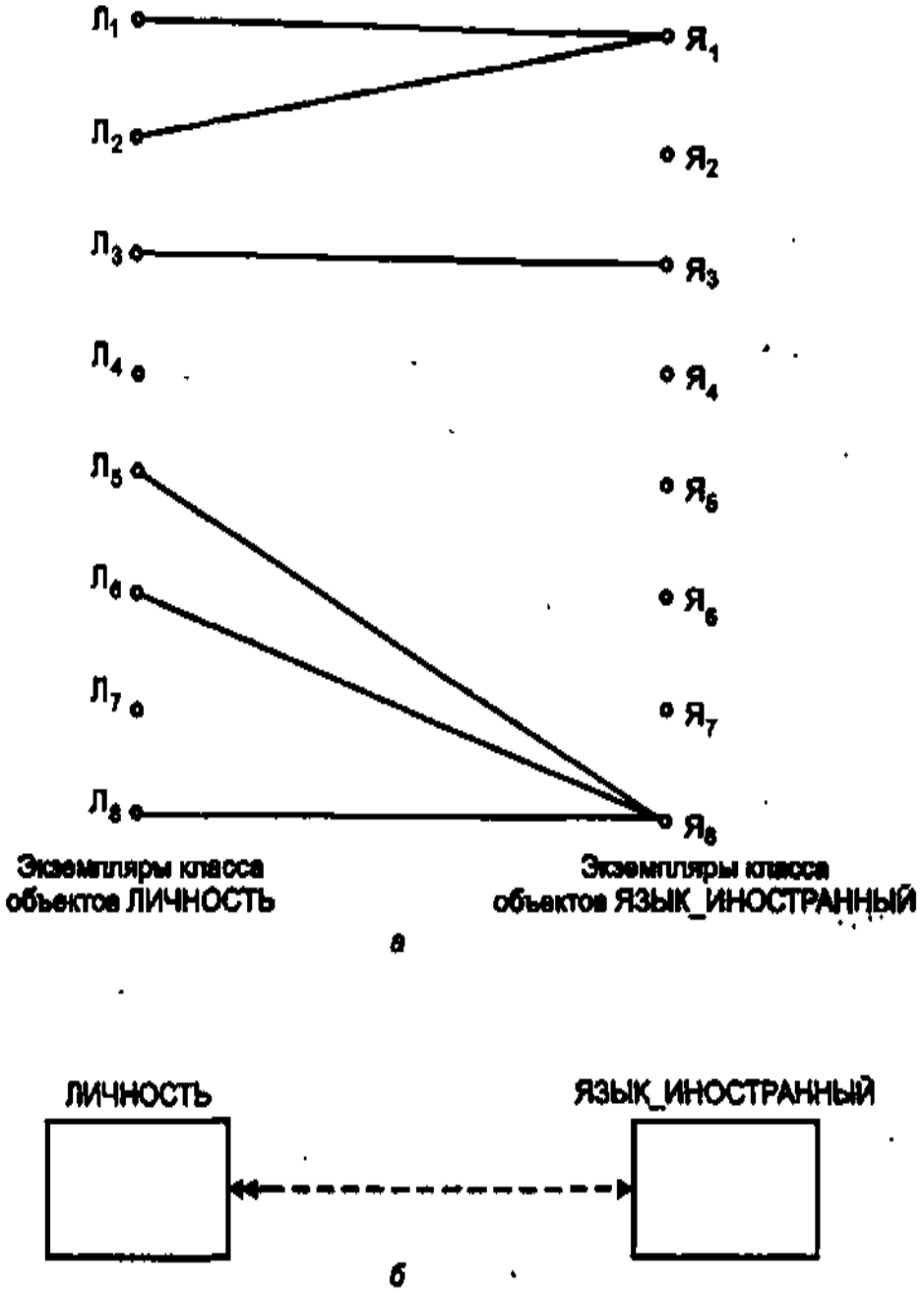


Рис. 2.17. Диаграмма ER-экземпляров *(а)* и ER-типов *(б)*.
Вариант 1 - необязательное членство с обоих концов связи

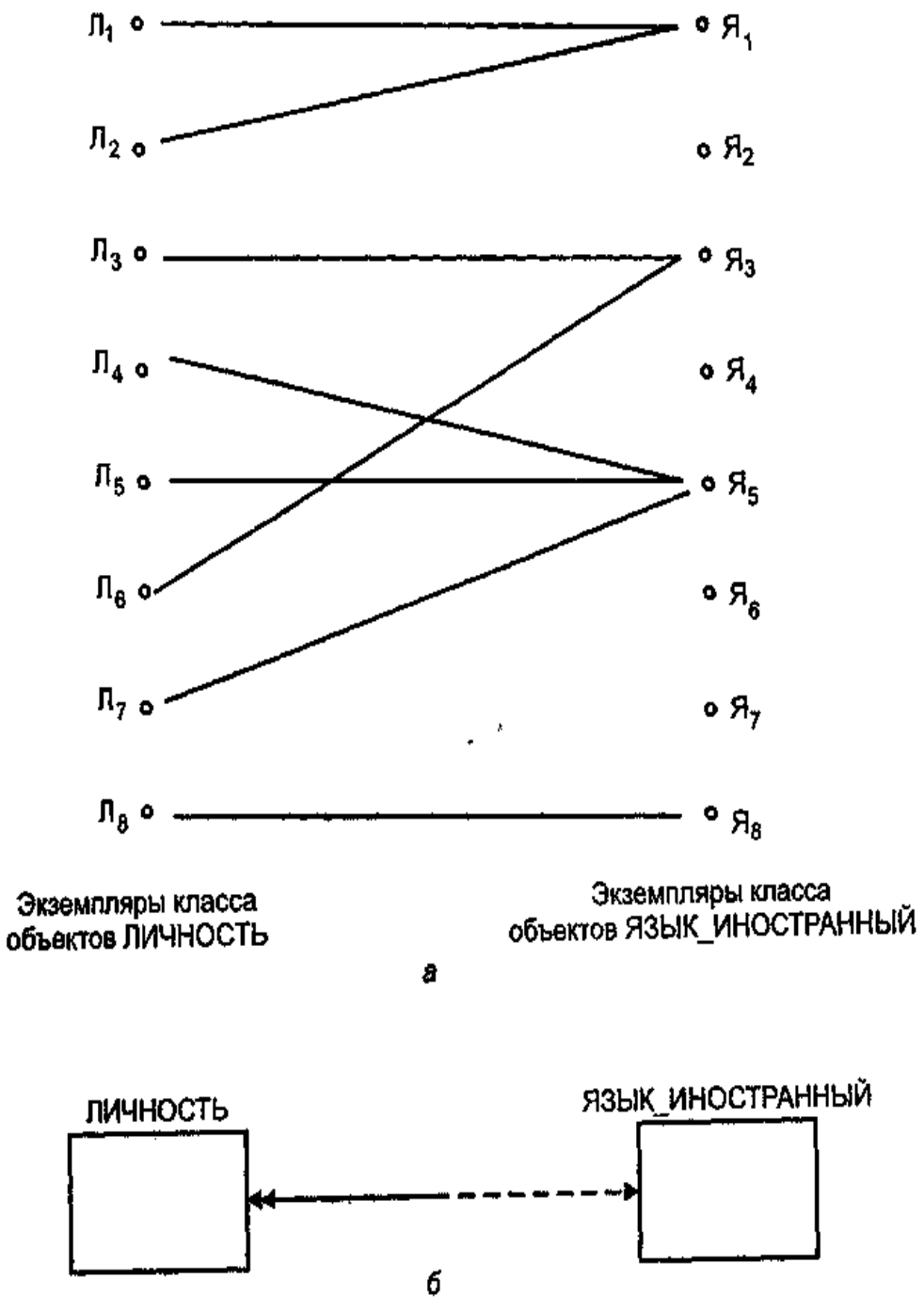


Рис. 2.18. Диаграмма ER-экземпляров *(а)* и ER-типов (б).
Вариант 2 - связь 1:M и необязательное членство с одного конца связи

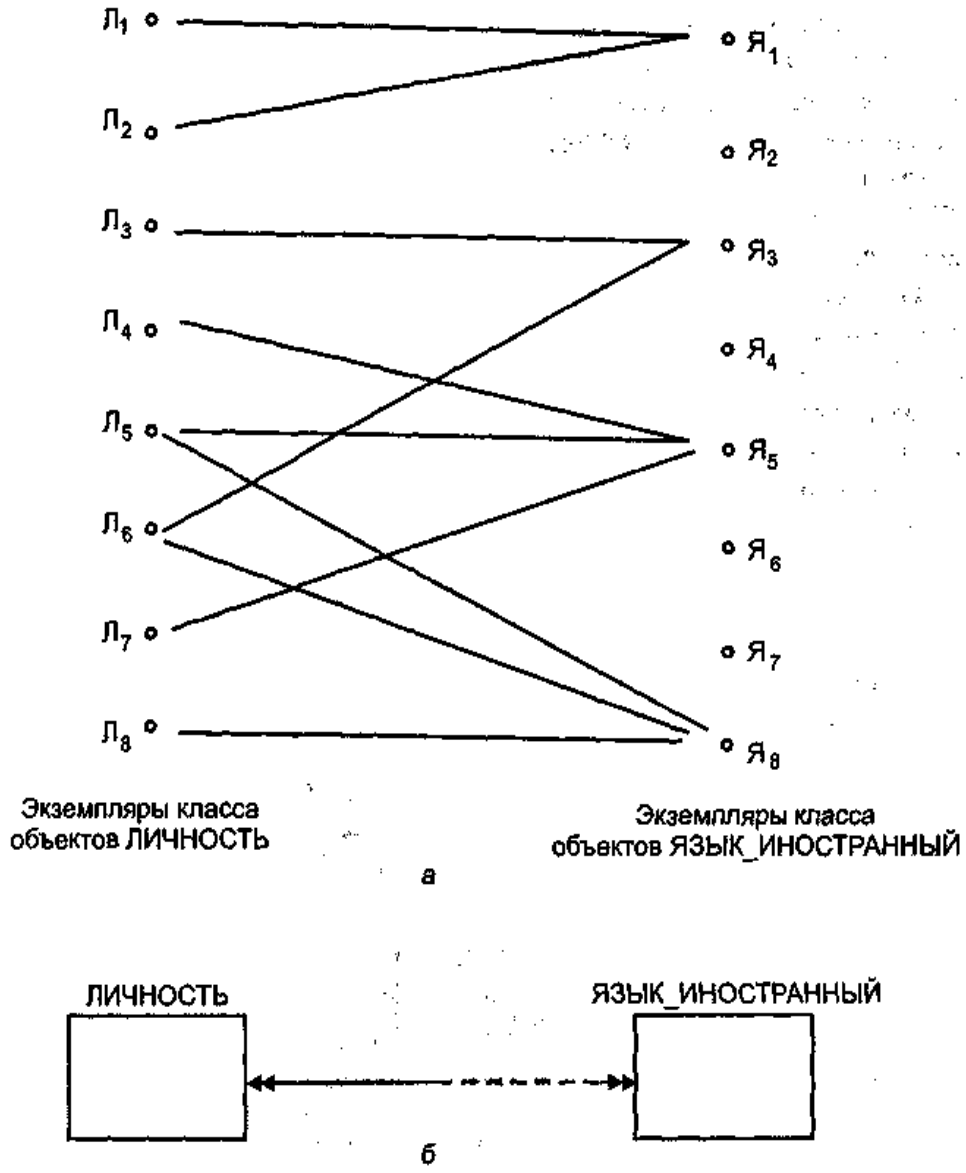


Рис. 2.19. Диаграмма ER-экземпляров (а) и ER-типов (б).
Вариант 3 - связь M:M и необязательное членство с одного конца связи

Предположим, что предметной областью является некоторый лингвистический институт, в котором каждый из сотрудников обязательно знает несколько иностранных языков, и по каждому из известных науке языков в этом институте имеется хотя бы один специалист, владеющий им. В этом случае связь между объектами будет M:M и класс принадлежности обеих сущностей является обязательным.

Примеры возможных ситуаций можно было бы продолжить, но суть уже ясна. Характер связи между объектами будет зависеть от особенностей предметной области. Например, если в вузе имеется экстернат и студент может обучаться по индивидуальному графику, то класс принадлежности объекта

СТУДЕНТ в связи с объектом ГРУППА будет необязательным, в противном случае он будет обязательным.

Отображение альтернативной связи. Возможны ситуации, когда в связи участвует один из нескольких возможных классов объектов. Например, если организация имеет центральный офис и филиалы, то служащий может работать либо в центре, либо в филиале, но не там и там одновременно. Назовем такие связи *альтернативными.* На схеме альтернативные связи будут объединяться скобкой (рис. 2.20).

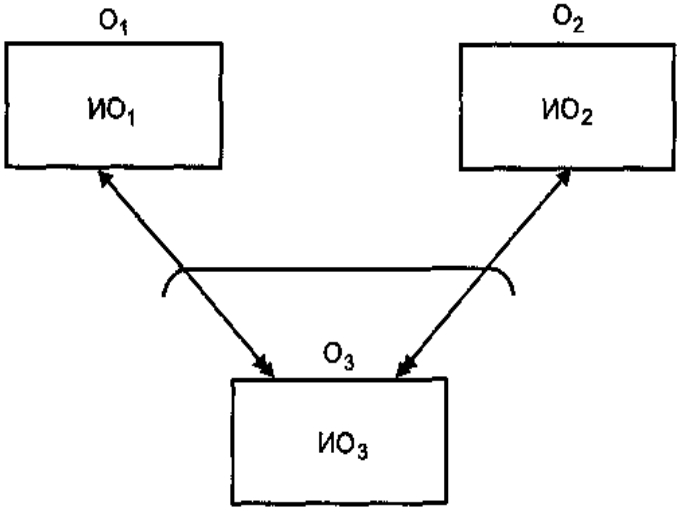


Рис. 2.20. Изображение альтернативной связи

2.2.8. Сложные объекты

Изображение обобщенных объектов. *Обобщенным* называется объект, в котором явным образом выделены подклассы.

Разбиение класса на подклассы осуществляется по какому-то признаку (свойству). Свойство, по которому проводится разбиение класса на подклассы, называется *дискриминатором.* Например, подклассы ВОЕННООБЯЗАННЫЕ и НЕВОЕННООБЯЗАННЫЕ выделяются в зависимости от значения свойства «Отношение к воинской обязанности»; подклассы СТУДЕНТЫ, АСПИРАНТЫ, ДОКТОРАНТЫ, ДОВУЗ выделяются в зависимости от значения свойства «Вид обучения».

Для обозначения подкласса в схеме будем использовать треугольник, связанный с обозначением свойства, по которому проводится разбиение на подклассы. Широкая часть треугольника направлена в сторону родового объекта, острый угол, к которому присоединены обозначения свойств, присущих данному подклассу, - в сторону видового (рис. 2.21, *а).*

На рис. 2.21, *б* изображен фрагмент инфологической модели, отражающий обобщенный объект ЛИЧНОСТЬ для высшего учебного заведения. Для него выделено несколько категорий объектов: СОТРУДНИК, УЧАЩИЙСЯ, СТУДЕНТ, АСПИРАНТ.

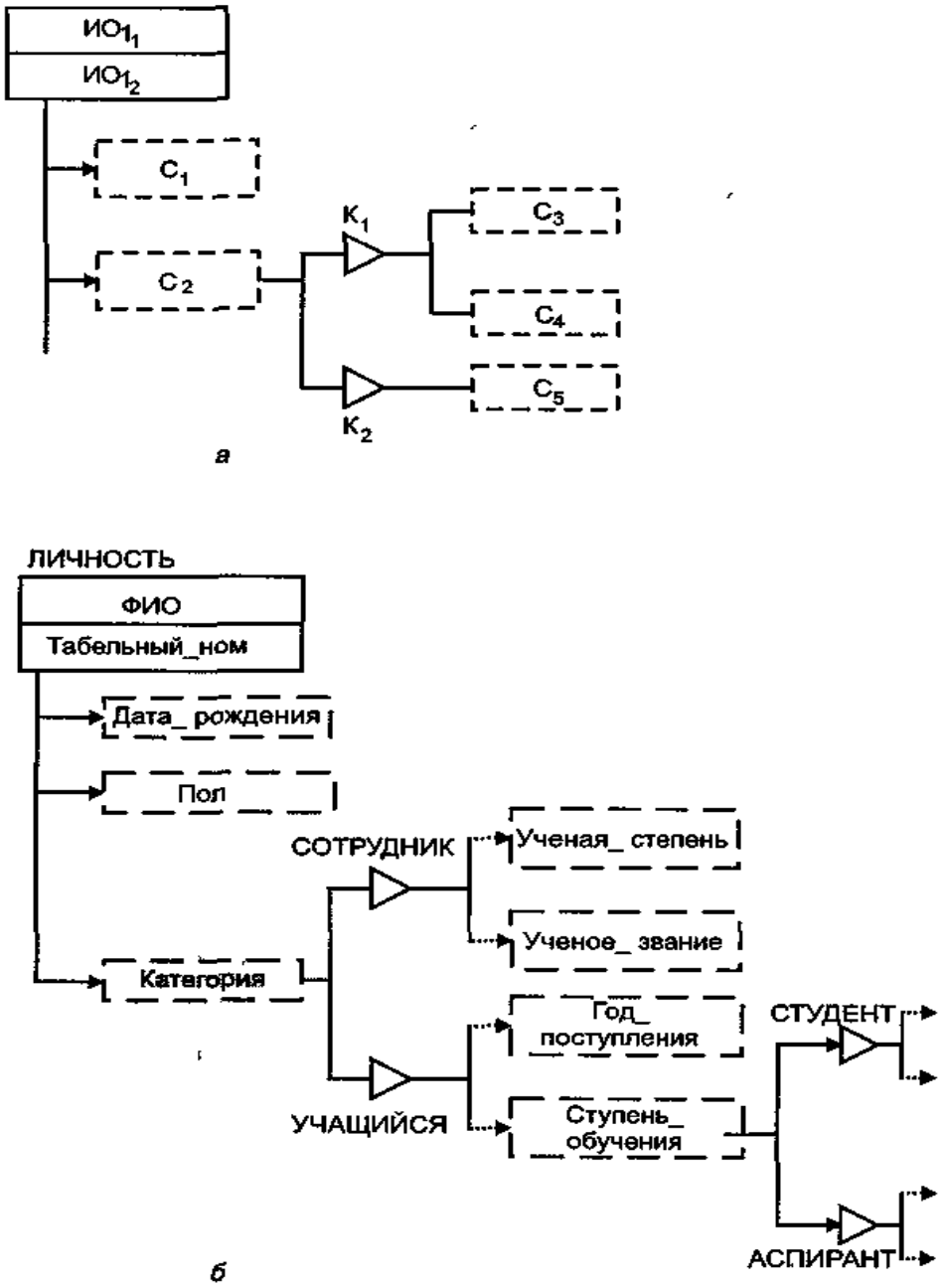


Рис. 2.21. Изображение обобщенного объекта:
а - условные обозначения; б - пример

Естественно, что классификация может быть многоуровневой. Так, в рассматриваемом примере обобщенный объект ЛИЧНОСТЬ может быть разбит на два подкласса: СОТРУДНИК и УЧАЩИЙСЯ. СОТРУДНИКИ, в свою очередь, могут быть классифицированы на ПРОФЕССОРСКО-ПРЕПОДАВАТЕЛЬСКИЙ СОСТАВ, АДМИНИСТРАЦИЯ и т.д.

Кроме того, подклассы в совокупности могут составлять исходный класс (полный класс), а могут представлять лишь часть ее (неполный класс). Если при описании предметной области возникает необходимость отобразить эту информацию, то для полного класса будем изображать двойную линию, перечеркивающую линию, идущую от дискриминатора; если класс неполный,

то будем изображать одинарную линию, перечеркивающую линию, идущую от дискриминатора.

Подкласс, как и класс, является совокупностью однотипных объектов. Отображать ли ту или иную сущность в виде отдельного класса или подкласса в составе обобщенного объекта - зависит от проектировщика. Изображение в виде обобщенного объекта является более информативным и, как следствие, дает больший выбор при принятии решений на стадии построения даталогической модели.

При использовании обобщенного объекта связи между объектами могут идти как к знаку всего обобщенного объекта, если объекты всех подклассов участвуют в данной связи, так и к знаку отдельного подкласса, если связь относится только к данному подклассу.

Обобщенный объект следует вводить в модель в том случае, когда нужно подчеркнуть общность и различие категорий объектов, входящих в один класс, или в случае, если объекты разных подклассов участвуют в разных связях.

Информация о пересекающихся классах. Выделенные в предметной области классы объектов могут быть как *пересекающимися,* так и *непересекающимися*[*]. Для отображения этих сведений в мифологической модели можно использовать граф пересечений, вершины которого соответствуют классам (подклассам) объектов, а ребра связывают пару вершин лишь в том случае, если соответствующие классы объектов являются пересекающимися. Для отображения степени пересечения можно воспользоваться взвешенным графом. При этом вес вершины будет обозначать мощность соответствующего множества объектов, а вес ребра - мощность множества, являющегося пересечением множеств, связанных этим ребром (рис. 2.22).

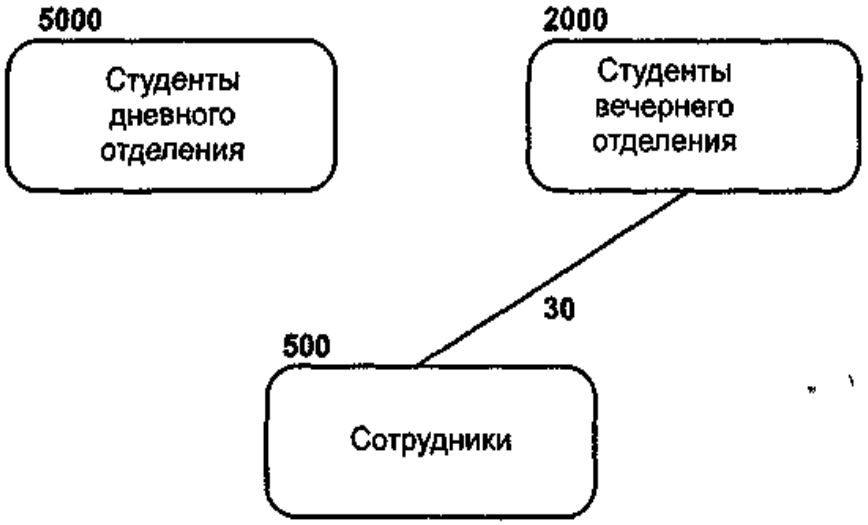


Рис. 2.22. Фрагмент графа пересечений

[*] Другое название - взаимно исключающие подклассы (когда объект может входить в один из подклассов, но не в несколько одновременно).

Эту же информацию можно отразить и в табличной форме (табл. 2.3).

Таблица 2.3

Класс 1	Класс 2	Размер пересечения

Изображение агрегированных объектов. Как отмечалось выше, агрегированные объекты (АО) соответствуют обычно какому-либо процессу, в который оказываются вовлеченными другие объекты. Для отображения агрегированного объекта в ER-модели будем использовать следующие условные обозначения: сам агрегированный объект будем изображать ромбом, рядом с которым указывается имя соответствующего агрегированного объекта. Ромб связывается с условными обозначениями тех объектов, которые образуют этот агрегированный объект. Свойства агрегированного объекта изображаются так же, как и для простого объекта (рис. 2.23, а).

В качестве примера агрегированного объекта из рассматриваемой предметной области «Учебный процесс» изобразим объект СДАЧА_ ЭКЗАМЕНА (рис. 2.23, б).

Изображение составных объектов. Для их отображения в ER-модели обычно не используются какие-либо специальные условные обозначения. Связь между составным объектом и составляющими его объектами отображается так же, как это было описано выше для простых объектов. Например, ГРУППА состоит из СТУДЕНТОВ, и это будет отображено просто как связь 1: М между этими объектами.

2.2.9. Рекомендации по построению базовой ER-модели

В ER-модели должно быть отображено все, о чем идет речь в данной предметной области (во входных документах, в выходных документах и т.п.). После построения полной ER-модели необходимо определить состав хранимых показателей. Переход от ER-модели к даталогической модели должен проводиться только для хранимых показателей.

При построении ER-модели необходимо ответить на вопросы:

1.Что следует считать самостоятельным объектом, а что - свойством другого объекта?

2.Когда следует делить класс на подклассы?

Резюмируя вышесказанное, можно дать следующие рекомендации. В качестве самостоятельного объекта в ER-модели следует изображать сущности:

- имеющие более одного идентификатора;
- для которых фиксируются какие-либо их свойства;
- которые участвуют более чем в одной связи.

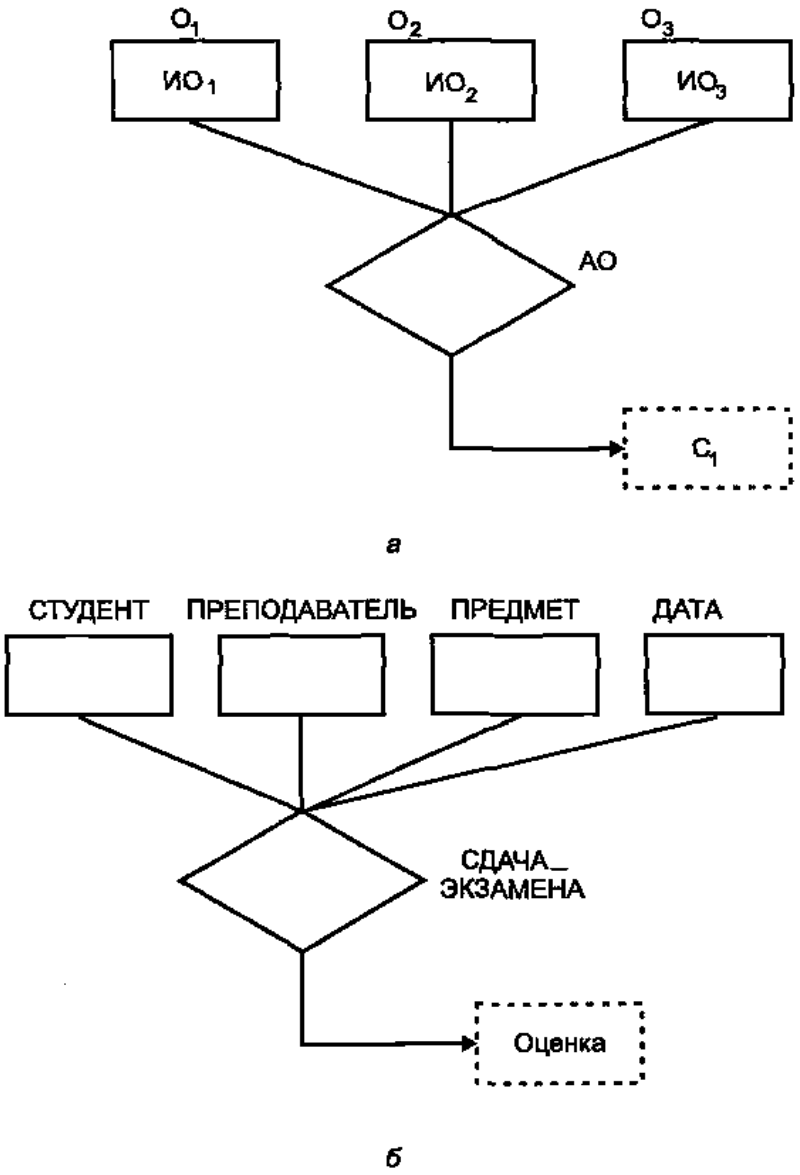


Рис. 2.23. Изображение агрегированного объекта:
а - условные обозначения; *б* - пример

При возникновении сомнений лучше принять решение о создании самостоятельного объекта, так как это в дальнейшем потребует меньших переделок модели.

Количественные характеристики всегда являются свойствами какого-либо объекта, и никогда - самостоятельными объектами. Количественные характеристики также практически никогда не являются идентификаторами объекта и не входят в состав идентификаторов.

При изображении предметной области нужно стремиться отобразить информацию как можно более детально, поскольку в дальнейшем это даст возможность принять более обоснованные решения при проектировании структуры базы данных. Так, например, если «Адрес», «ФИО» являются составными характеристиками, то желательно это отразить в ER-модели.

При решении вопроса о том, что следует отображать в качестве обобщенного объекта, приходится выбирать между двумя крайними вариантами: надо ли простой объект представить как обобщенный и надо ли два либо несколько самостоятельных объектов объединить в обобщенный объект.

Обобщенный объект следует вводить в модель в том случае, когда нужно подчеркнуть общность и различие категорий объектов, входящих в один класс, или в случае, если объекты разных подклассов участвуют в разных связях. Так, например, если для сотрудников мужского и женского пола фиксируются одни и те же свойства, эти объекты участвуют в одних и тех же связях, то не следует выделять соответствующие подклассы. Если же для студентов мужского пола фиксируются сведения о воинской обязанности, информация о том, прошли ли они срочную службу, занимаются ли они на военной кафедре и т.п., а для студенток эта информация не фиксируется, то разбивать класс объектов СТУДЕНТ на подклассы следует.

Естественно, что каждый подкласс может быть изображен в ER-модели как самостоятельный объект, а не как подкласс какого-то родового класса. Для того чтобы иметь больше информации о предметной области и (часто) сократить число элементов (свойств, связей) в ER-модели, в большинстве случаев лучше объединять подклассы в класс.

Одну и ту же ситуацию в предметной области можно представить в ER-модели разными способами. На рис. 2.24 изображены фрагменты ER-модели, отображающие факт знания сотрудником иностранных языков. Каждый из изображенных вариантов при всем их различии является правильным.

Если ЯЗЫК не будет использоваться ни в каких связях, то возможно использование каждого из приведенных вариантов. Если такой уверенности нет, то вариант (б) лучше не использовать.

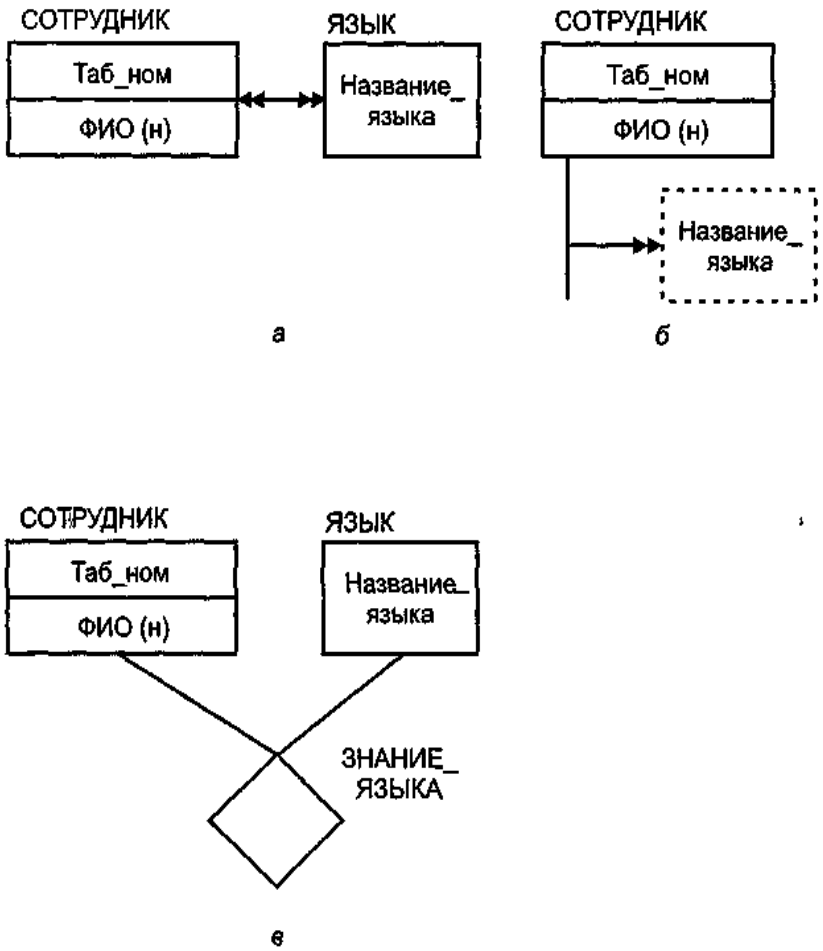


Рис. 2.24. Варианты изображения связи СОТРУДНИК - ЯЗЫК:
а - путем указания связи между объектами;
б - с использованием множественного свойства;
в - с применением интегрированного объекта

Хотелось бы обратить внимание на некоторые наиболее часто допускаемые ошибки в процессе моделирования. Одной из таких ошибок является изображение зависимых друг от друга свойств в виде самостоятельных, не связанных друг с другом. Так, на рис. 2.25, *а* изображен неправильный вариант отображения информации о степени владения сотрудником тем или иным иностранным языком. Кроме правильного варианта, изображенного на рис. 2.25, *б,* возможен вариант, аналогичный изображенному на рис. 2.24, *б*), где «Степень_владения» будет свойством агрегированного объекта ЗНАНИЕ_ЯЗЫКА.

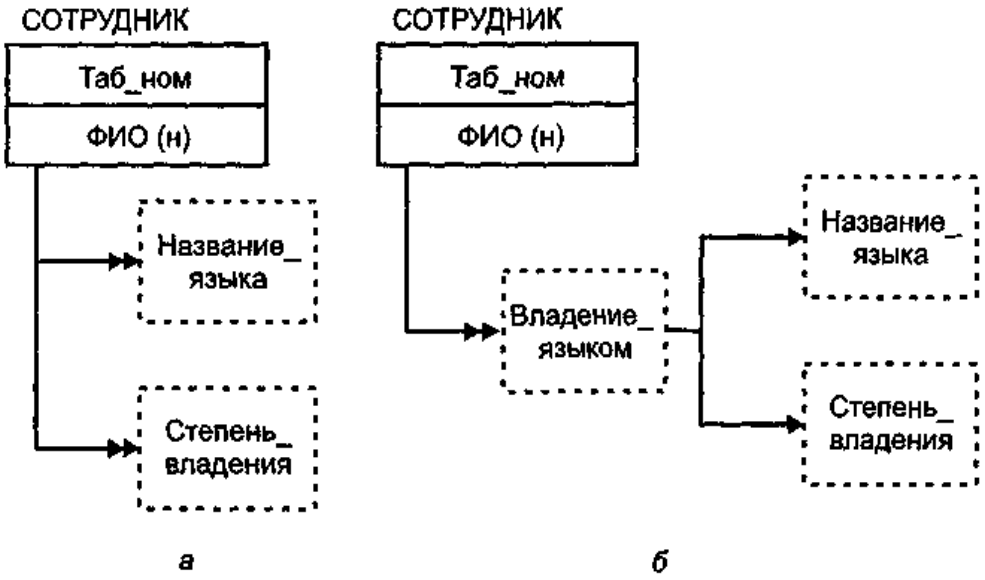


Рис. 2.25. Изображение информации о владении иностранными языками в базовой ER-модели: ошибочный *(а)* и правильный *(б)* варианты

Следует напомнить, что нельзя устанавливать связь между свойством одного объекта и другим объектом или его свойством. Нельзя также непосредственно связывать между собой агрегированные объекты. Связь устанавливается на уровне объектов. Подклассы могут участвовать в связях, так же как и классы.

2.3. Сравнение методик построения ER-моделей

ER-модели широко используются в практике создания баз данных. Причем они применяются как при ручном, так и при автоматизированном проектировании. Чаще всего для представления ER-модели используются графические языки. Мы в основном будем рассматривать именно их. Изобразительные средства и методики графического представления ER-моделей, используемые в разных системах автоматизации проектирования, а также в разных литературных источниках, несколько отличаются друг от друга.

Далее мы рассмотрим особенности представления ER-моделей на примере нескольких наиболее известных систем автоматизации проектирования (CASE-систем): ProKit*WORKBENCH, Design/IDEF, CASE Oracle (Designer/2000), Power Designer (новое название, используемое для последних версий системы S-Designor), ERWin, SILVERRUN, ERStudio и др., а также методологий, изложенных в некоторых литературных источниках.

Можно выделить целый ряд признаков для сравнения систем моделирования предметных областей и последующего проектирования автоматизированных информационных систем. Часть из них одинаково применима как для ручного, так и для автоматизированного проектирования, другие

признаки имеют значение только при использовании автоматизированных инструментальных средств проектирования.

Сначала рассмотрим аспекты, присущие обоим способам создания ИС.

Прежде всего остановимся на различиях в использовании изобразительных средств. Можно выделить несколько категорий различий в изображении ER-моделей.

2.3.1. Несущественные различия в использовании условных обозначений

Для обозначения простого объекта в разных системах используются прямоугольники, блоки с закругленными углами, овалы и т.д.

Наблюдаются различия в способе изображения характера связей между объектами (использование «стрелок», «лапок», «точек» и т.п. для отображения «множественного» конца связи). Такого рода различия можно продолжить. Они не накладывают никакого отпечатка на методологию построения концептуальной модели и алгоритм последующего перехода к даталогической модели.

Желательно, чтобы используемые обозначения были интуитивно понятными, излишне не загромождали модель, были просты в изображении. Часто предпочтения разработчиков в использовании тех или иных обозначений определяются просто привычкой. По возможности следует стремиться к использованию стандартизированных или широко распространенных обозначений.

В некоторых методиках при изображении связи между объектами в разъеме отображающей ее линии предлагается изображать ромб и внутри него или рядом с ним писать название связи (модель Чена) (рис. 2.26).

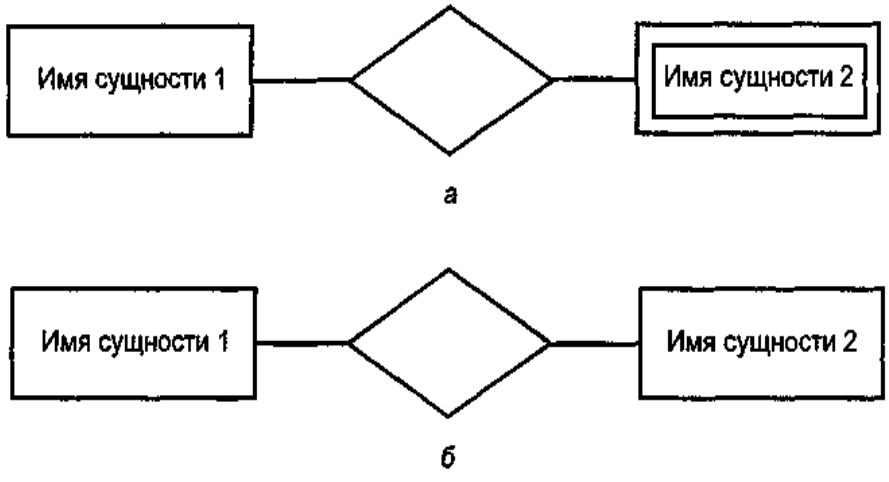


Рис. 2.26. Диаграмма Чена.
Условные обозначения:
а - зависимая сущность; *б* - независимая

Другие способы изображения не требуют использования ромбов или требуют их изображения не всегда, а только при наличии определенных ситуаций

в отображаемой предметной области. Если модель допускает связывание только пары объектов (т.е. поддерживает изображение только бинарных связей) и не допускает фиксирование дополнительных свойств для связей, то введение дополнительного обозначения только загромождает модель и не дает никаких дополнительных преимуществ.

Поскольку в разных методологиях проектирования используются разные условные обозначения для отображения одного и того же явления в предметной области (т.е. фактически наблюдается синонимия в графическом языке описания предметной области), то некоторые системы автоматизации проектирования, например ProKit*WORKBENCH, предоставляют пользователю возможность выбрать из множества допустимых обозначений те, которые ему больше нравятся или более привычны. В этой системе, к примеру, для обозначения вида связей между объектами могут использоваться условные обозначения, представленные на рис. 2.27, а сама модель может изображаться в стиле Чена, Бахмана или в реляционном виде. Наиболее предпочтительный для пользователей системы стиль изображения должен быть выбран перед построением модели.

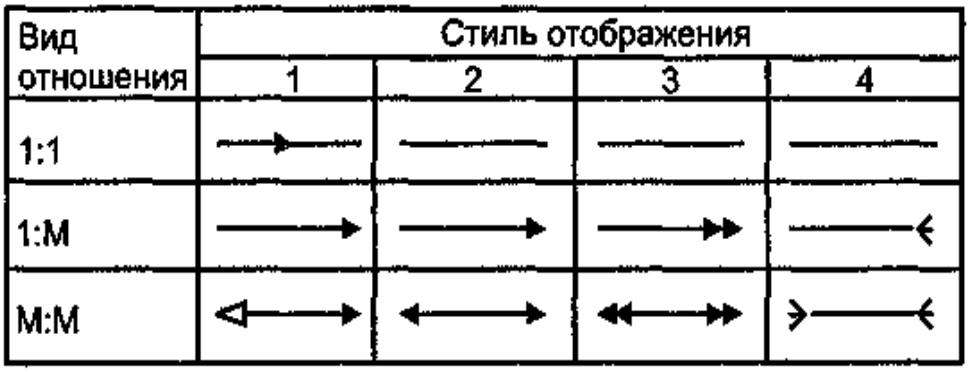


Рис. 2.27. Изображение связей между объектами в системе ProKit*WORKBENCH

Рассмотрим некоторые из указанных выше различий более подробно.

Для отображения обязательности вхождения объектов в связь («класс принадлежности») в разных системах моделирования используются разные условные обозначения. Так, в CASE Oracle класс принадлежности передается следующим образом: с той стороны связи, с которой элемент может не обязательно входить в связь, используется пунктирная линия, а там, где членство обязательное, - сплошная линия. С учетом класса членства возможны типы отношений, представленные на рис. 2.28.

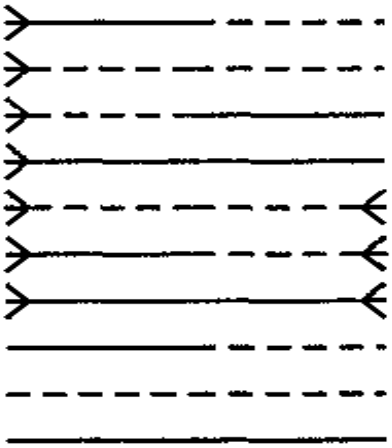


Рис. 2.28. Варианты изображения типа связи
и класса членства в CASE Oracle

В [15] для отображения класса принадлежности используется маленький прямоугольник, который рисуется внутри блока, изображающего объект. Если класс принадлежности объекта в связи является обязательным, то внутри этого прямоугольника ставится точка. Если класс принадлежности необязательный, то точка ставится вне блока или вообще не ставится (рис. 2.29).

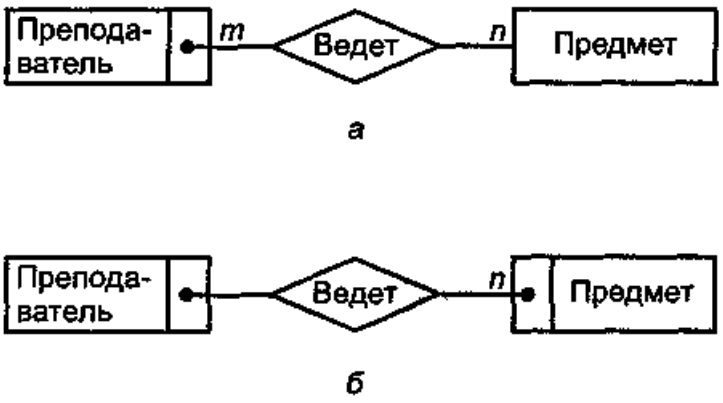


Рис. 2.29. Варианты изображения типа связи и класса членства
(Г. Джексон): обязательный класс членства с одной стороны (а)
и с обеих сторон (*б*)

Используемые в CASE Oracle обозначения более удобны, так как если объект участвует в большом количестве связей, то дополнительные прямоугольники с точками становится неудобно располагать на рисунке.

В Design/IDEF1X характер принадлежности в связи изображается, как показано на рис. 2.30. Точка на конце линии обозначает множественную связь. Если около точки не стоит никакой буквы, то это означает нуль, один или более объектов в связи; P(positive) - один или много, Z (zero) - нуль или один, N (целое положительное число) - мощность связи в точности равна некоторому числу.

Наблюдаемые в Design/IDEFIX отличия от базовой и других рассмотренных выше моделей являются более существенными, чем просто различия в используемых условных обозначениях, поскольку это не только другой способ обозначения, но и другое «множество» возможных сочетаний «тип связи»-«класс принадлежности». Поэтому эти различия следует отнести ко второму типу различий - существенным различиям, влияющим на процесс моделирования предметной области.

В Power Desinger используются обозначения, представленные на рис. 2.31.

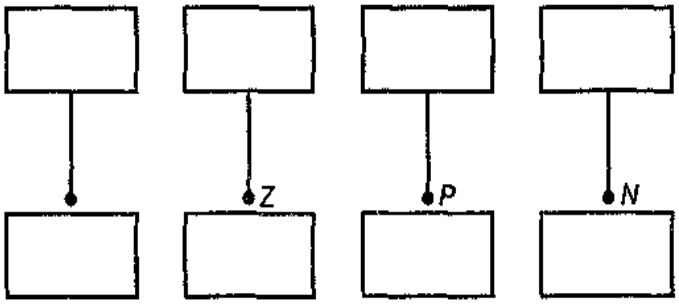


Рис. 2.30. Изображение класса членства в Design/IDEF1X

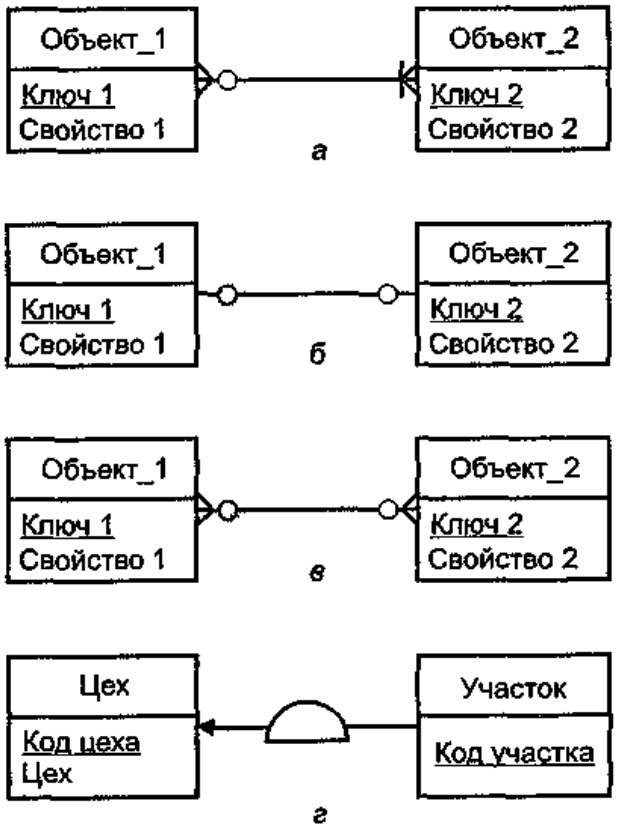


Рис. 2.31. Обозначения, использующиеся в Power Desinger:
а - связь M:M, связь от Объекта_1 к Объекту_2 необязательная;
б - связь 1:1, обязательный класс членства с обоих концов;
в - связь M:M, класс членства обязательный с обоих концов;
г - изображение зависимой по идентификации сущности

В Paradigma Plus для характеристики связи используется термин *multiplicity* (множественность): Каждый конец связи имеет аннотацию, обозначающую множественность. Различают следующие разновидности множественности: one (один), many (много), optional (необязательный), one or more (один или больше), zero or more (нуль или больше).

На первый взгляд кажется, что подходы, используемые в Design/IDEF и Paradigma Plus, схожи между собой. Но в них есть существенное различие: в Paradigma Plus, так же как и в нашей базовой модели, тип связи и кардинальность относятся к каждой стороне связи (т.е. характер связи задается в прямом и обратном направлении), в Design/IDEF - только в прямом. Поэтому в Design/IDEF нельзя выразить, например, связь M:M, в которой с одной стороны связи наблюдается обязательный класс принадлежности, а с другой - необязательный.

Подход, принятый в базовой модели для отображения характера связи, представляется более продуктивным (экономичным и наглядным), так как для всего множества возможных сочетаний (с учетом направления связи их может быть 16) используется всего четыре условных обозначения.

Чаще всего для обозначения какого-либо явления в предметной области в конкретной методологии используется одно определенное обозначение. Но, как отмечалось выше, некоторые CASE-средства позволяют пользователю выбрать ту нотацию языка, которая является для него наиболее привычной.

Свойства объекта иногда не отображаются на той же схеме, что сами объекты и связи между ними, а перечень и описания этих свойств приводятся отдельно. Часто описание свойств представляют в табличной или иной аналитической форме, а не в графическом виде.

Поскольку рассматриваемые различия не являются существенными, то легко выполнить преобразование из одной формы представления в другую, что и позволяют автоматически делать многие CASE-средства.

2.3.2. Различия в использовании и изобразительных средств, приводящие к изменениям в методике построения модели

Некоторые различия, также связанные со способом изображения тех или иных ситуаций, являются более существенными, приводящими к различиям в методике построения модели, в адекватности изображения ПО и т.п. Например, в системе CASE Oracle обобщенный объект изображается путем вложения блоков, обозначающих видовые объекты, внутрь блока, изображающего родовой объект. На рис. 2.32 показано изображение объекта ЛИЧНОСТЬ, рассмотренного выше (*см.* рис. 2.21), в условных обозначениях, используемых в CASE Oracle.

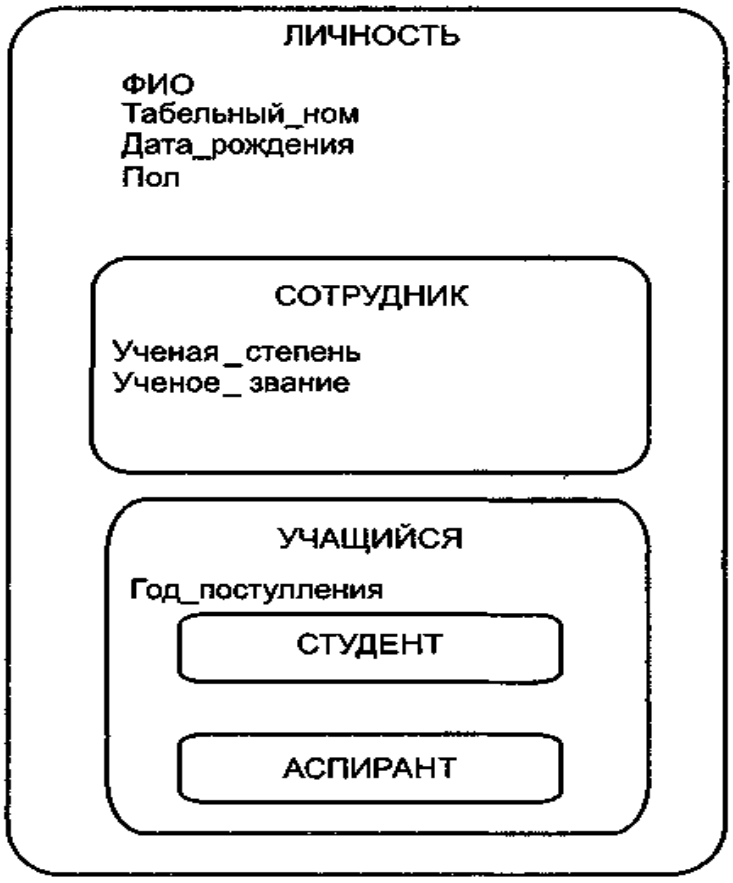


Рис. 2.32. Изображение обобщенного объекта в CASE Oracle

Как следует из сравнения рисунков, изображения обобщенных объектов в сравниваемых методиках различаются не только по форме представления. Так, если объект классифицируется по разным признакам, то при использовании первого способа (*см.* рис. 2.21) наглядно видно, по какому признаку осуществляется классификация. Второй же способ изображения (*см.* рис. 2.32) не обеспечивает этого.

Другими словами, предложенный в базовой модели способ изображения обобщенных объектов является семантически более содержательным, информативным, емким.

На рис. 2.33 изображен тот же обобщенный объект ЛИЧНОСТЬ с использованием синтаксиса IDEF1X. По своей семантике этот способ изображения ближе к предложенному нами базовому способу изображения ИЛМ. Разница заключается в том, что для сущностей-категорий и общих сущностей в IDEF1X используются одинаковые обозначения сущности (правда, в большинстве случаев родовой объект является независимой, а видовой - всегда зависимой от идентификации сущностью). Атрибут, по которому проводится разбиение (дискриминатор), выносится из состава атрибутов обобщенного объекта и становится именем, располагаемым рядом со значком дискриминатора. И хотя по форме представления изображения обобщенных объектов в нашей базовой модели и IDEF1X сильно различаются, но по мощности они идентичны, и различия между этими моделями можно отнести к рассмотренным выше различиям первого класса.

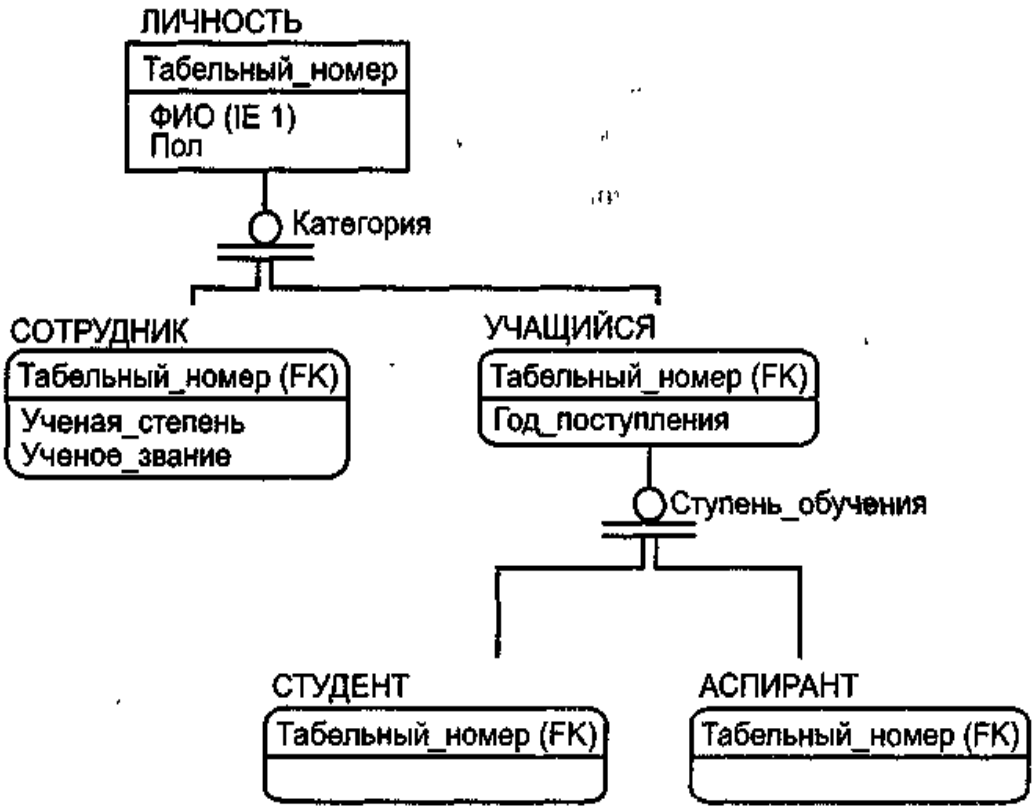


Рис. 2.33. Изображение обобщенного объекта ЛИЧНОСТЬ в IDEF1X

Предложенный нами в базовой модели способ обозначения кажется более четким:

1) он фиксирует внимание на том, что обобщенный объект представляет собой множество достаточно однородных объектов, имеющих общие свойства, а дискриминатор хотя и играет специфическую роль, но является одним из свойств объекта;

2) обобщенный объект изображается как единая сущность, а не совокупность множества отдельных объектов.

С методологической точки зрения способ изображения в IDEF1X акцентирует внимание на том, что видовые объекты - это самостоятельные объекты. В нашей же модели (как и в CASE Oracle), наоборот, констатируется, что обобщенный объект, включающий подклассы, является тем не менее объединяющей сущностью.

Близким к IDEF1X относительно методологии отображения обобщенных объектов является способ их изображения в CASE-средстве Vantage Team Builder. В нем для обозначения признака, по которому проводится разбиение на подклассы, используется ромб, который соединен как с супертипом, так и с каждым из подтипов. Линия, соединяющая ромб с супертипом, перечеркивается.

2.3.3. Пространственное размещение элементов ER-модели

ER-модель даже для небольшой и несложной предметной области включает в себя описание значительного числа компонентов и связей между ними. При этом встает проблема наглядности общей схемы. Эта проблема поразному решается при ручном и автоматизированном построении инфологической модели. В автоматизированных системах чаще всего строится единое изображение ER-модели и используется прием масштабирования, когда, уменьшая или увеличивая масштаб изображения на экране, можно посмотреть как всю схему, так и отдельный ее фрагмент.

Многие CASE-средства позволяют выделять фрагменты из общей схемы и работать с ними как с самостоятельными компонентами модели, а также проводить объединение отдельных фрагментов в единую схему.

Различные приемы используются и для того, чтобы уменьшить число пересечений линий на схеме. Так, в системе ProKit*WORKBENCH для этих целей допускается дублирование изображения объекта и размещение дубля рядом с тем объектом, с которым его надо связать. Чтобы показать, что это не новый объект, а дубликат уже изображенного в модели объекта, используется какое-либо условное обозначение, например у соответствующих блоков, отражающих дубликат, отчеркивается уголок.

При ручном проектировании изобразить всю ER-модель в виде единой схемы обычно не представляется возможным. В этом случае можно порекомендовать следующий прием: изобразить и описать каждый объект самостоятельно, присвоить каждому объекту короткий код. Используя эти кодовые обозначения, для каждого объекта указать его связи с другими объектами в виде отдельных схем. При этом нужно не упускать из виду, что, несмотря на такую дефрагментацию при изображении, модель является единой и связанной.

Многие CASE-системы позволяют выводить на экран информацию с разной степенью детализации (например, только названия сущностей, либо сущности и все их свойства, либо только ключевые атрибуты и т.п.). Такие возможности совсем не присущи ручным способам проектирования (хотя могут использоваться как методологический прием: проектирование сначала выполняется с минимальной степенью детализации, а затем проводится последовательное повышение степени детализации - обычный прием в структурном проектировании).

2.3.4. Отсутствующие возможности

Некоторые возможности, имеющиеся в одних системах или методиках, отсутствуют в других. В этих случаях возможны различные варианты:

- для изображения ситуации, имеющей место в предметной области, используются возможности, предоставляемые данной методологией, но это требует применения определенных приемов, часто несколько искусственных, для их представления;

- ситуация просто не отображается в модели.

Например, во многих системах инфологического моделирования предполагается, что свойства у объекта могут быть только единичными., В этом случае каждое множественное свойство следует представлять как самостоятельный объект и изображать связь между этим вновь введенным объектом и исходным объектом.

Так, в IDEF1X, как и в других широко известных CASE-системах, свойства объекта могут быть только единичные и всегда определенные (не условные). Если свойство может отсутствовать у каких-либо объектов, то надо выделять отдельные сущности, например СЛУЖАЩИЙ_ШТАТНЫЙ с атрибутом «Оклад» и ПОЧАСОВИК, не имеющий такого атрибута. Это приведет к необходимости выделения большого числа объектов и связей в ИЛМ, к снижению наглядности модели. Например, отдельные экземпляры объекта ЛИЧНОСТЬ могут иметь или не иметь ученое звание, ученую степень, год окончания вуза и много других признаков. По каждому из этих признаков придется выделять подклассы (либо не фиксировать в модели, являются ли эти свойства условными).

Некоторые методики не вводят агрегированный объект как самостоятельную категорию. В таком случае сущность, соответствующая агрегированному объекту, изображается как простой объект; при этом пользователь должен предварительно определить идентификатор этого объекта (что является далеко не тривиальной задачей) и его свойства. Далее изображение будет зависеть от того, могут ли в используемой нотации быть изображены только бинарные связи либо с объектом можно связать несколько разных объектов.

Кроме указанных сложностей при определении идентификатора агрегированной сущности могут возникнуть и проблемы при переходе от ИЛМ к даталогической модели.

Если модель допускает изображение только двоичных (бинарных) связей, то проектировщик должен преобразовать n-арную связь в совокупность бинарных.

Связь типа «арк», использованная в базовой модели и имеющаяся в CASE Oracle, отсутствует в большинстве других рассматриваемых в данном учебнике систем. Это делает неудобным отображение тех ситуаций, где следовало бы использовать эту возможность (чаще всего при этом приходится вводить лишнюю сущность).

Если методика построения модели не предполагает фиксацию класса принадлежности объекта в связи, то эта информация будет просто потеряна.

Ни в одной из известных нам систем моделирования нет понятия *составное свойство.* Поэтому при моделировании предметной области проектировщик должен либо каждый из составляющих элементов составного свойства изобразить как отдельные самостоятельные свойства, либо изобразить это свойство без разделения на составляющие. И в том, и в другом случае часть информации о ПО будет утеряна: в первом случае мы не увидим,

что элементы составного свойства являются логически единым целым, а во втором - не увидим его состава.

В некоторых CASE-системах имеет место ситуация, когда какая-то конструкция допускается в системе как промежуточная. Например, в IDEF1X и CASE Oracle связь M:M допускается как так называемое неспецифическое отношение. Его наличие разрешается на ранних стадиях разработки проекта, но в дальнейшем оно должно быть заменено на специфическое отношение (т.е. отношение типа 1:M). Это достигается посредством введения в модель дополнительной третьей сущности и соединения с ней исходных сущностей связью типа 1:M (другими словами, одна связь M:M заменяется на две 1:M). Это является недостатком подобных систем, так как, во-первых, не все целевые СУБД требуют такого преобразования (некоторые системы поддерживают отношение M:M в явном виде) и, во-вторых, если такое преобразование все-таки потребуется, система автоматизации проектирования вполне могла бы выполнить его автоматически на этапе даталогического проектирования. Даже если выполняется ручное проектирование, то указанное преобразование должно выполняться проектировщиком на стадии даталогического проектирования, а не при описании предметной области. Кроме того, при рассматриваемом преобразовании на стадии инфологического проектирования в IDEF вводится новая категория сущностей - сущности пересечения, или ассоциативные сущности. Введение новых сущностей влечет за собой введение в ER-модель и дополнительных связей. Все это вместе взятое усложняет и без того нелегкую задачу инфологического проектирования.

В предметной области могут быть сущности, идентификаторы которых являются зависимыми от идентификатора какого-то другого объекта. Например, если участки на предприятии нумеруются в пределах цеха, то идентификатор участка будет составным, включающим в себя код цеха и код участка. В инфологической модели можно ограничиться указанием этого составного идентификатора. Некоторые методики построения ER-моделей (например, IDEF1X, ProKit*WORKBENCH) предусматривают введение особых видов сущностей и особых видов отношений для отображения подобных ситуаций. Так, в IDEF1X сущность, для идентификации которой надо рассматривать ее отношение с другими сущностями, называется *зависимой от идентификатора сущностью,* и для ее изображения используется блок с закругленными углами (в отличие от сущности, не зависимой от идентификации, для обозначения которой используются прямоугольники). Для связи объектов, один из которых нужен для полной идентификации другого, вводится понятие *идентифицирующего отношения. Дня* него также вводится свое условное обозначение. В IDEF1X для идентифицирующего отношения используется сплошная линия, а для неидентифицирующего - пунктирная.

Способ идентификации отражает не особенности предметной области, а языковые характеристики, а именно - способ именования объектов. Но лингвистические отношения тоже являются частью ИЛМ, и они могут быть отражены в ER-модели.

Если модель не использует в явном виде указание на способ идентификации объекта, то, как указывалось выше, эту ситуацию нужно отразить просто путем соответствующего использования идентификаторов. Так, например, если на предприятии участки нумеруются в пределах цеха, то в качестве идентификатора объекта ЦЕХ следует использовать «Код_цеха*Код_участка» (рис. 2.34). Следует обратить внимание на то, что прямоугольник, соответствующий зависимой по идентификации сущности, в этом случае на сектора не делится.

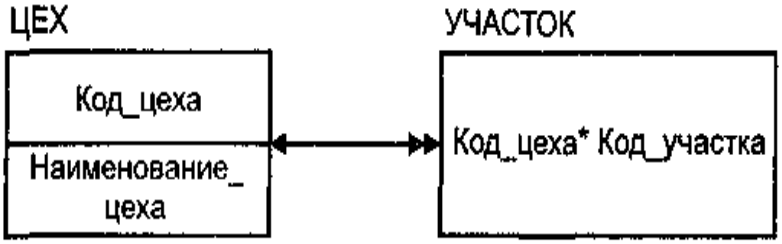


Рис. 2.34. Изображение зависимой по идентификации сущности
в случае отсутствия специальных обозначений

Как отмечалось выше, ИЛМ включает в свой состав много разнообразных компонентов. Методологии моделирования и конкретные системы различаются *полнотой и широтой охвата характеристик,* отражаемых при описании предметной области. Так, некоторые системы предусматривают описание запросов (в частности, ключей поиска), количественных характеристик классов объектов и запросов, ограничений целостности и т.д., другие - нет. Указанные описания иногда бывают объединены с ER-моделью (например, в ProKit*WORKBENCH) или оформляются как отдельные самостоятельные компоненты.

2.3.5. Различия в классификации объектов и отношений между ними

Как отмечалось выше при рассмотрении принципов инфологического моделирования, понятия «объект», «свойство», «отношение» являются относительными. Так, в предложенной нами базовой инфологической модели выделяются разные виды объектов: простые, составные, агрегированные, обобщенные. В некоторых системах моделирования такой классификации объектов нет и вместо этого используются разновидности отношений.

И тот, и другой подход имеет, право на существование. Принципиальной разницы, влекущей за собой какие-то серьезные последствия, в сравниваемых подходах нет.

2.3.6. Терминологические различия

Кроме различия в изображении тех или иных сущностей, в теории инфологического моделирования наблюдается расхождение в используемой терминологии. Например, в CASE Oracle родовой объект называется супер-

тип (syper-type), а видовой - подтип (sub-type). Таких различий в терминологии можно привести много, но их фиксация не является сейчас нашей целью. Хотя обратить внимание на такие различия нужно, чтобы при использовании конкретной системы не возникло ошибочное понимание происходящего. Так, в некоторых CASE-системах (например, Design/IDEF, PowerDesigner и др.) используется непривычная для теории проектирования БД трактовка понятия физического проектирования: под физическим проектированием в них понимается проектирование (описание) структуры БД для конкретной целевой СУБД.

2.3.7. Соглашения по именованию элементов ER-модели

При описании базовой модели мы использовали только имена классов объектов, идентификаторов и свойств объектов.

Правила задания имен сущностей различаются в разных методологиях моделирования ПО. Эти различия касаются того, что обязательно, а что не обязательно именовать, где размещать имена и т.п.

Многие методики требуют обязательного именования связей. Поскольку связи являются двусторонними, то наименование связи будет меняться в зависимости от того, с какой стороны ее рассматривать. Поэтому часто в модели предлагается указывать оба эти названия (например, в системах CASE Oracle, ProKit*WORKBENCH). В Design/IDEF1X обязательно надо указывать имя связи от «родителя» к «ребенку», имя связи в обратном направлении также можно указать, но это не является обязательным.

Для того чтобы было понятно, к какому из направлений связи какое название относится, принимают определенные соглашения о том, как располагать эти названия на схемах. Например, сверху линии помещать название, относящееся к левой стороне связи, а под линией - к правой; или разделять эти имена наклонной чертой (первым помещается имя от «родителя» к «ребенку»).

Наличие большого числа обозначений и подписей загромождает модель. Кроме того, само присвоение названий часто представляет некоторую трудность, что увеличивает трудоемкость инфологического моделирования. Поэтому в тех случаях, когда это не приводит к двусмысленности и неясностям и если это позволяет система, можно рекомендовать не использовать дополнительные особые обозначения (помимо линии, соединяющей эти объекты) и имена для связей.

Для CASE-средств важной является также связь вопросов именования элементов ER-модели и соответствующих им элементов даталогической модели. Так как ER-модель описывает предметную область и используется всеми категориями пользователей для обеспечения ее однозначного понимания, то желательно, чтобы имена давались на естественном национальном языке. Но в дальнейшем ER-модель используется для генерации описания структуры БД, а конкретные СУБД имеют разные ограничения на задание имен элементов БД. Хорошим решением этой проблемы является наличие в CASE-

средстве возможности автоматически преобразовывать имена, использованные в ER-модели, в соответствии с ограничениями целевой СУБД. К сожалению, далеко не все CASE-средства обладают такими возможностями. Некоторые CASE-средства позволяют при создании ER-модели каждому ее элементу давать несколько имен (одно - для использования в концептуальной модели, другое - в даталогической (или, как она называется во многих современных CASE-средствах, -физической) модели и др.). Это, конечно, лучше, чем полное отсутствие возможности использовать разные имена, но не вполне соответствует сути подхода, заключающейся в том, что по исходному описанию могут генерироваться проекты для разных целевых систем, так как ограничения на допустимые имена в каждой из этих систем могут быть разными.

К сожалению, многие CASE-средства не только не обеспечивают, но даже и не контролируют правильность задания имен в полученной схеме БД. Это надо иметь в виду при построении ER-модели. При использовании некоторых CASE-средств возникают проблемы с использованием русского языка.

2.3.8. Дополнительные характеристики CASE-средств

ER-модель является сердцевиной систем проектирования, но, естественно, не единственным ее компонентом. Для CASE-систем* кроме общей характеристики используемой в них методологии ER-моделирования, существенной как для автоматизированного, так и для ручного проектирования, необходимо учитывать и специфические критерии, связанные с реализацией функций автоматизированного проектирования.

Ниже перечислены некоторые характеристики, которые следует учитывать при сравнении CASE-систем:

- число и перечень поддерживаемых целевых СУБД (а если не ограничиваться только рассмотрением вопросов проектирования БД, как делается в этом учебнике, то не только СУБД, но и других инструментальных средств);
- поддержка разных технологий организации БД (локальные, распределенные);
- поддержка коллективной работы при проектировании (в том числе управление правами различных пользователей, ведение общего репозитория, возможность консолидации фрагментов моделей, созданных разными пользователями);
- построение концептуальной (ER-модели) по описанию структуры существующей БД - ремоделирование (реверс-инжиниринг);
- поддерживаемые виды моделей (кроме концептуальной (ER-модели));

* CASE-системы являются сравнительно новым направлением в информационных технологиях. Так, первая версия CASE-инструментария фирмы Oracle появилась в 1989 г.

- автоматизируемые функции проектирования и степень их автоматизации;
- качество проектных решений; жесткость решений (возможность выбрать из нескольких альтернативных решений наиболее подходящее для данной ситуации, возможность ручного вмешательства в процесс проектирования на разных его этапах);
- надежность работы;
- документирование проекта;
- открытость системы (возможность стыковки с другими средствами автоматизации проектирования);
- удобство графического редактора;
- количественные ограничения (общее число объектов, число уровней вложенности для обобщенного объекта и др.);
- возможность автоматически оценивать объем памяти для проектируемой БД;
- возможность автоматической генерации хранимых процедур, процедур ввода данных и т.п.;
- наличие средств для моделирования хранилищ данных;
- требования к ресурсам компьютера;
- операционная среда;
- стоимость системы.

При оценке автоматизированных систем проектирования необходимо сравнивать не только язык модели и алгоритмы проектирования, но и саму реализацию: удобство интерфейса, степень автоматизации проектных решений, удобство использования и полноту справочной системы и другие характеристики именно программной реализации системы. Так, например, Help в Design/IDEF имеет множество недостатков: помощь неконтекстная; если действие по созданию какого-либо объекта не закончено (открыто какое-то окно), то помощь вообще недоступна; подписи, поясняющие назначение кнопок меню, высвечиваются только при нажатии на них; если кнопка является неактивной (т.е. ее использование невозможно в данной ситуации), то и узнать назначение этой кнопки нельзя; далеко не все термины, сокращения, обозначения можно найти в справке; практически не представлены методические вопросы, нет рекомендаций по моделированию. В противоположность этому можно привести пример ERWin: система включает не только Help в привычном понимании, но и обширные методические материалы с большим количеством примеров.

Многие автоматизированные средства проектирования позволяют просматривать модель с разной степенью детализации: только обозначения сущностей и связей между ними или сущность+ключи, или сущность+ключи+внешние_ключи, или сущность+все_атрибуты. Наличие таких возможностей создает существенные удобства, особенно при создании больших и сложных моделей.

Другой важной характеристикой CASE-средств является возможность получать различные подмодели из общей модели и, наоборот, обеспечивать интеграцию различных фрагментов в единую модель. Эти возможности могут быть полезны не только при коллективной разработке проекта несколькими разработчиками: очень удобно при обсуждении модели с конечными пользователями для каждого из них предоставлять только ту часть модели, которая имеет для него интерес; декомпозиция модели с возможностью корректной интеграции разных подсистем облегчает процесс проектирования.

Еще одним критерием для сравнения CASE-средств является степень проверки правильности построенных моделей. Следует обратить внимание, что ни одна система автоматизации проектирования, насколько бы развитой она ни была, не может гарантировать соответствия построенной концептуальной модели реалиям предметной области. Это определяется только квалификацией разработчиков, их пониманием предметной области и умением адекватно отобразить ее в модели. Но наличие средств проверки моделей может помочь устранить ошибки, связанные в основном с невнимательностью, такие, как отсутствие идентификатора у сущности, отсутствие связи объекта с другими объектами, неправильное задание имен, отсутствие в модели информации, необходимой/полезной при дальнейшем проектировании (объемные характеристики для классов объектов и связей между ними и т.п.), противоречия в модели (особенно существенно при коллективной разработке) и др.

Даже при использовании одной и той же методологии, как, например, в Design/IDEF и ERWin (обе используют методологию IDEF1X), способ их машинной реализации оказывается разным. Так, например, в Design/IDEF при описании атрибута указывается, что он является дискриминатором, при этом на схеме автоматически выводится знак дискриминатора и его имя. В ERWin не выделяется свойство, по которому проводится разбиение, и «дискриминатор» автоматически не именуется. Это менее удобно. Хотя можно потом выделить соответствующую связь для редактирования и в описании супертипа указать, какой атрибут является дискриминатором. Тогда название этого атрибута повторяется рядом со знаком дискриминатора.

Анализируя CASE-средства, мы обращали внимание только на те характеристики, которые оказывают непосредственное влияние на проектирование структуры базы данных. Однако функции CASE-средств этим не ограничиваются. Многие системы позволяют задавать в модели ограничения целостности и генерируют программы (триггеры, хранимые процедуры), проверяющие эти ограничения при эксплуатации БД. Кроме того, CASE-средства могут генерировать программы ведения БД.

Многие CASE-средства позволяют экспортировать модели в другие системы и, наоборот, импортировать их из других систем.

Отметим некоторые частные особенности конкретных CASE-средств.

Design/IDEF (версия 3.5). При описании атрибута в этой системе указывается, является ли он первичным ключом (Primary Key - PK) или инвертированным входом (Invertory Entry - IE). Подход, при котором эти вопросы

приходится решать на стадии моделирования предметной области, представляется неудачным по следующим причинам:

- «ключ» - понятие, относящееся к реляционной модели, а не к предметной области;
- вопрос, что выбрать в качестве ключа, нужно решать на стадии проектирования даталогической модели, а не инфологического моделирования, и желательно - автоматически;
- специалисту, строящему ER-модель, необходимо знать, что такое «ключ», «внешний ключ», «альтернативный ключ», уметь их определять и выбирать, что не так и просто, особенно если речь идет о составном ключе (как отмечалось выше, желательно активное участие специалистов предметных областей в разработке ER-модели, и требовать от всех них знания таких специфических вопросов теории баз данных нежелательно);
- для нескольких атрибутов можно указать, что он является PK. На самом деле это означает, что каждый из таким образом обозначенных атрибутов является элементом составного ключа, а не самим ключом, что в аспекте реляционной теории принципиально важно различать;
- термин «инвертированный вход» вообще не очень подходит для ER-модели, так как определяет способ организации хранения данных (для «инвертированных входов» строится индекс);
- система не проверяет ошибочные с точки зрения реляционной модели определения атрибутов. Так, можно объявить какой-то атрибут одновременно и первичным, и вероятным ключом или объявить один атрибут первичным ключом (т.е. ключ простой) и его же включить в состав составного альтернативного ключа, или можно не задать длину атрибута, и при этом генерируется описание таблицы БД без указания длины поля.

Если в Design/IDEF вы изобразили объект и установили с ним связь 1:M от другого объекта, то сразу в подчиненный объект мигрирует ключ из родительского объекта. Это же закладывается автоматически и в алгоритм проектирования, что далеко не всегда целесообразно (то же самое наблюдается и в ERwin, и во многих других системах).

Недостатком такого подхода является то, что вместо мифологического моделирования на самом деле как бы сразу происходит проектирование отношения (таблицы БД). Но, во-первых, связь 1:M может быть отражена в БД разными путями (*см.* описание алгоритма в главе 3), а во-вторых, на уровне ER-модели происходит дублирование информации (наличие связи между сущностями отображается дважды: линией на схеме и повторением ключа родительской сущности).

В системе имеется целый ряд ошибок, связанных с преобразованием ER-модели в целевую схему БД. Так, например, если изобразить связь M:M (которая называется неспецифической), то схема (на SQL) сформируется, но с

ошибками (ни в одну из таблиц не включается ключ связанной таблицы, не создается таблица связи, но в то же время создается внешний ключ, которому не соответствует первичный ключ). В документации по CASE-системам, в которых связь M:M является неспецифической, сказано, что на поздних стадиях моделирования их надо преобразовать в специфические. Но если забыть это сделать, то, как сказано выше, описание на SQL сформируется и никаких сообщений об ошибках выдано не будет. Кроме того, наблюдается неполное соответствие типов данных, определяемых CASE-средством, реальному списку типов данных, поддерживаемому целевой СУБД.

Как отмечалось выше, разные целевые системы могут иметь разнообразные ограничения на допустимые имена используемых информационных единиц. В Design/IDEF имена полей в SQL-описании даются те, которыми в модели названы атрибуты (это могут быть, например, длинные русские слова или фразы с пробелами; в большинстве случаев задание таких имен недопустимо в целевых СУБД, но система не контролирует допустимость используемых имен информационных единиц).

ERWin (версия 2.6). В этой системе просматривать модель можно на разных уровнях: только объекты, объекты и свойства, связи с указанием кардинальности или без оной, с указанием ограничений целостности и без указания и др. Для того чтобы воспользоваться этой возможностью, нужно в меню «Option» отметить позицию «Show Display menu» и в появившемся меню выбрать нужные параметры отображения.

При связывании сущностей, так же как и в Design/IDEF, ключ родительской сущности всегда мигрирует в зависимую сущность в качестве неключевого атрибута (при построении модели родительской становится та сущность, на которую «кликнули» мышью первой).

Система накладывает следующие ограничения на размер модели: до 500 сущностей и до 1500 атрибутов.

При изображении подтипов используется иной подход, нежели в Design/IDEF1X, а именно: изображаются сначала все объекты, как родовой, так и видовые. Потом они соединяются соответствующей связью. В отличие от Design/IDEF1X в ERWin не выделяется свойство, по которому проводится разбиение, и дискриминатор первоначально никак не именуется. Но потом можно (щелкнув на значке дискриминатора, выйти в соответствующий редактор) указать, по какому атрибуту идет разделение, и имя этого атрибута появится около значка дискриминатора.

В ERWin версии 3.5 введены дополнительные возможности:

- поддержка многомерного моделирования;
- возможность оценивать объем таблиц и индексов в БД;
- словарь доменов;
- расширение опций для генерации баз данных;
- поддержка последних версий некоторых серверов баз данных и некоторые другие.

Редактор Design/IDEF интуитивно более понятен и привычен (например, чтобы перемещать линию, нужно щелкнуть по ней мышью и перемещать метки концов или середины линии; чтобы ввести новый атрибут в сущность, достаточно двойного щелчка по этой сущности). Но более привычно не всегда означает, что это безусловно лучше. Так, при перетаскивании линий в ERWin нельзя оторвать концы линий от границы знака сущностей, которые они связывают, что скорее является достоинством, чем недостатком.

S-Designor (новое название - PowerDesigner). Система включает в себя следующие модули:

- ProcessAnalyst - потоки данных и бизнес-процессы; позволяет создать функциональную модель информационной системы.
- DataArchitect - концептуальная и физическая модель; позволяет создавать концептуальную модель, основываясь на информации функциональной модели из ProcessAnalyst, с последующей генерацией физической модели для более чем 40 различных СУБД. Возможна обратная генерация физической модели из существующей БД.
- AppModeler - генерация приложений; позволяет генерировать отдельные объекты или целые приложения для PowerBulder, Visual Basic и ряда других инструментов.
- MetaWorks - поддержка коллективной работы.
- Начиная с версии 6.0 в PowerDesigner появился новый модуль WarehouseArchitect, предназначенный для проектирования специализированных информационных моделей для хранилищ данных, включая многомерные информационные модели. Помимо поддержки традиционных СУБД позволяет генерировать физические модели для специализированных серверов Sybase IQ и Red Brick.

Имеется объект Домен (Domain), который предназначен для определения типа данных. Посредством использования этого объекта можно определить допустимые значения для каждого из элементов данных. Для Домена можно указать тип данных, длину, допустимые значения и некоторые другие свойства.

С одним Доменом можно связывать несколько элементов данных. Возможность задания Доменов, таким образом, сокращает описание концептуальной модели и позволяет стандартизировать представление однотипных данных.

2.3.9. Использование графических ПП для изображения ER-моделей

При изображении ER-моделей можно воспользоваться различными графическими средствами. Так, широко распространенное средство Visio Professional (рис. 2.35) поддерживает множество нотаций ERD (Entity - Relationship Diagrams - диаграммы «сущность-связь»).

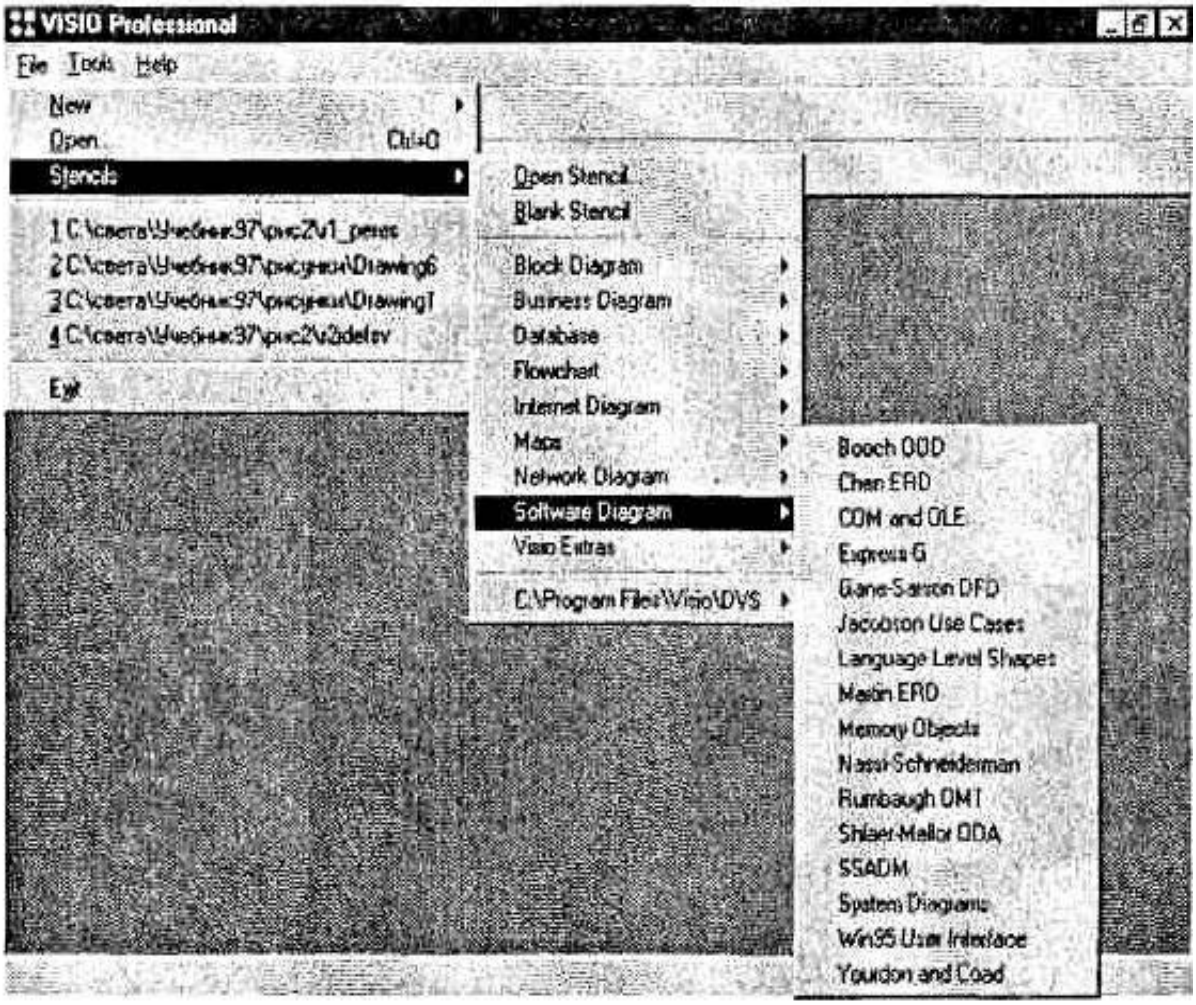


Рис. 2.35. Нотации, поддерживаемые в Visio Professional

В Microsoft Visio ER-модели можно рисовать, используя панель меню ***Stensils/Database/Entity Relationship*** (рис. 2.36).

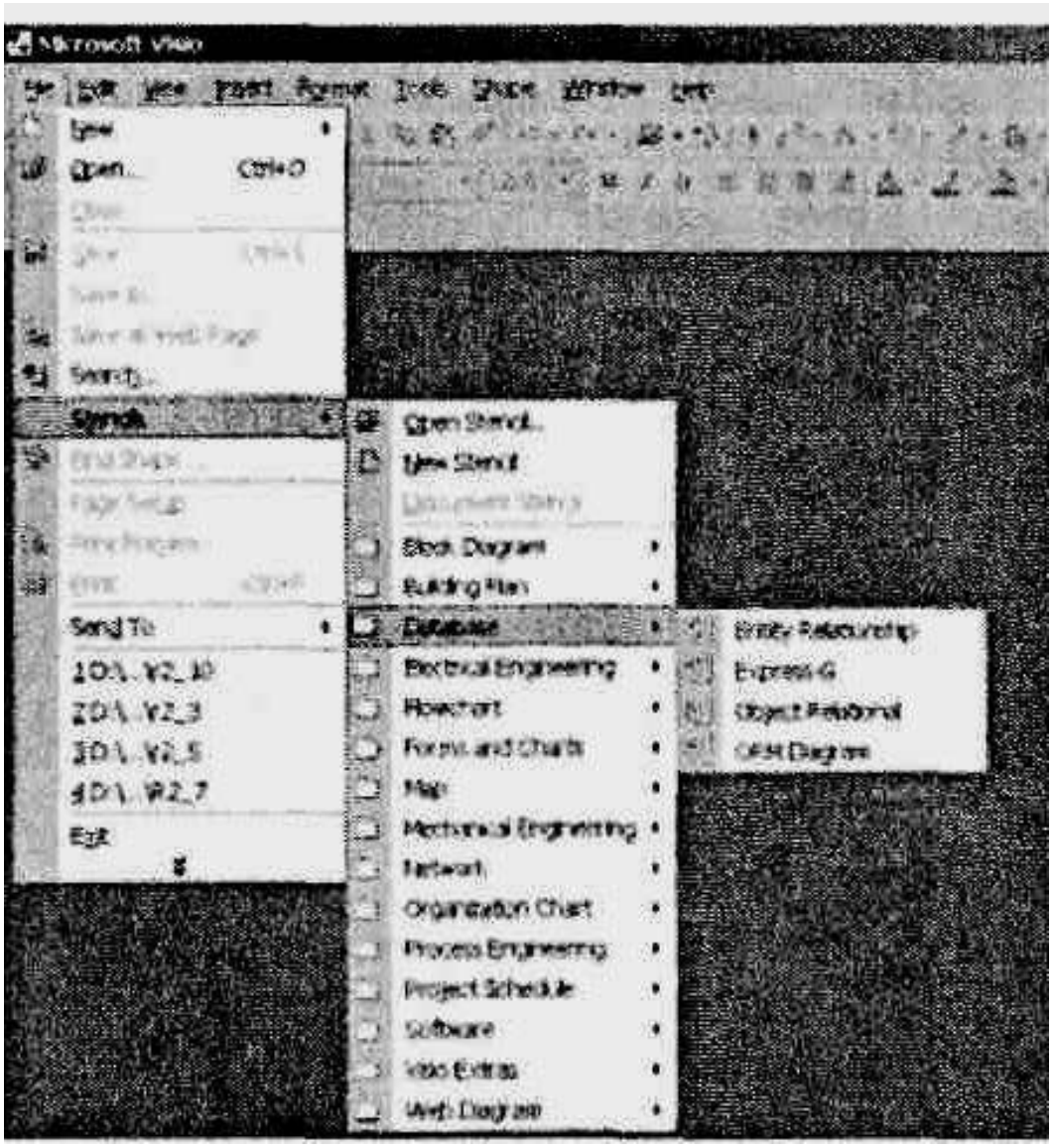


Рис. 2.36. Вид окна Microsoft Visio.
Меню **Stensils/Database/Entity Relationship**

Условные обозначения, используемые при этом (рис. 2.37), соответствуют методологии IDEF1X.

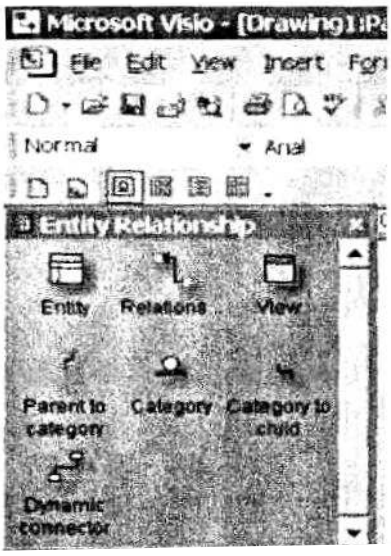


Рис. 2.37. Условные обозначения, используемые при построении ER-диаграмм (Microsoft Visio)

Позиции меню ***Stensils/Software*** (рис. 2.38) в Microsoft Visio соответствуют разновидностям моделей в UML. Эти модели напрямую для проектирования структуры базы данных не используются.

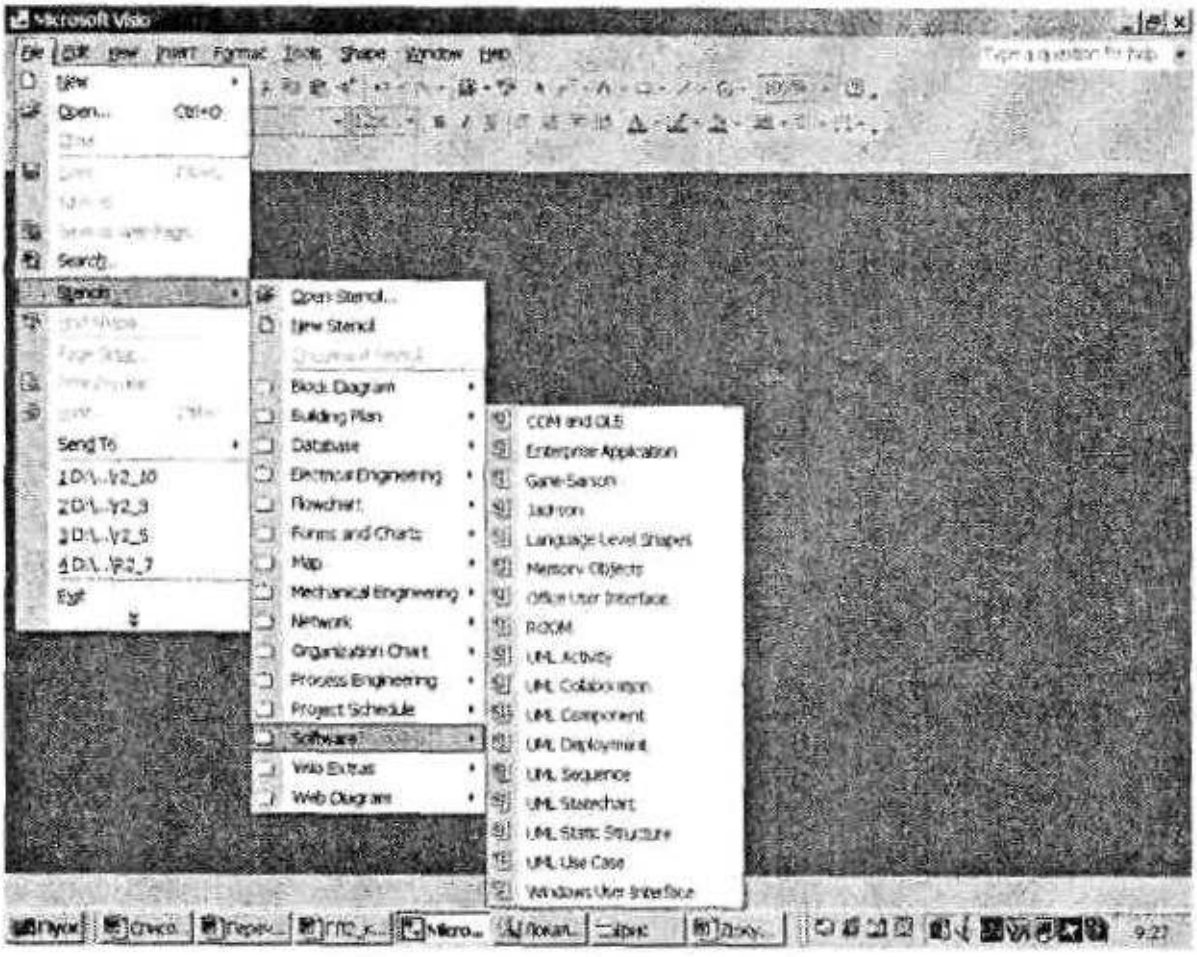


Рис. 2.38. Вид окна Microsoft Visio. Меню **Stensils/Software**

2.4. Особенности методологии построения ER-моделей

Как мы видели, выразительные возможности языковых средств представления ER-моделей в разных CASE-системах, а также «ручных» методиках моделирования отличаются друг от друга, и иногда очень существенно. Это, безусловно, накладывает отпечаток на методику построения ER-модели в каждой конкретной среде.

Принципиально важным является решение вопроса о том, что же отражает ER-модель. Во многих методологиях и документации по конкретным CASE-средствам считается, что ER-модель является концептуальной моделью БД. Основным отличием «базового» подхода, изложенного в учебнике, от многих методик, реализованных в современных CASE-средствах, является то, что в нем моделируется предметная область, тогда как в большинстве методологий отображается база данных (более того, именно реляционная БД). В предметной области нет понятия «ключ», «неспецифическое отношение» и т. п. (более того, эти понятия отсутствуют и в некоторых моделях данных, отличных от реляционных), которые вводятся во многих рассмотренных выше системах. Выбор ключа, кандидата ключа в излагаемой в учебнике методике проектирования переносится в «даталогическое проектирование», а в большинстве существующих методик решается на уровне концептуального моделирования.

В ERWin вводятся понятия *логический уровень* (что у нас называется ИЛМ (или концептуальная модель) и *физический уровень* (что соответствует ДЛМ), и считается, что одними и теми же средствами (в рамках одной модели) можно отразить и тот, и другой уровень. На самом деле это не совсем так, поскольку часть свойств, присущих конкретной СУБД, должна быть определена все-таки не в ER-модели, а при генерации схемы.

Кроме того, ни одна из анализируемых систем вообще не рассматривает проблему определения состава хранимых в БД данных (во многих из этих систем вообще все, что имеется в ER-модели, «переносится» в схему БД). Это служит еще одним косвенным подтверждением того, что ER-модель понимается, несмотря на декларации, все-таки как модель БД, а не модель предметной области.

В предлагаемой в данном учебнике методологии проектирования в ER-модели должны быть отображены все элементы, о которых идет речь в исследуемой предметной области. В процессе проектирования БД выполняется ряд шагов, в том числе и определение состава показателей, хранимых в базе данных. В базу данных могут переноситься не все атрибуты, присутствующие в ER-модели. Так, производные показатели часто не включаются в БД.

Большинство современных CASE-систем, таких, как Design/IDEF, ER-Win и др., не содержат блоков, осуществляющих определение состава атрибутов, подлежащих хранению в БД (хотя даже некоторые из более ранних систем автоматизации проектирования предусматривали выполнение таких шагов). Некоторые системы, например ERWin, хотя и не позволяют автоматически определять состав показателей, хранимых в БД, но дают возмож-

ность пометить атрибуты, которые присутствуют только в концептуальной модели.

В связи с этим ER-модель, подлежащая преобразованию в модель целевой БД, при использовании CASE-систем, в которых не только нельзя автоматически определять состав хранимых атрибутов, но и нельзя их хотя бы пометить, должна содержать только те данные, которые должны храниться в БД.

В таких ситуациях рекомендуется строить две ER-модели: первая будет отображать предметную область в целом, безотносительно к тому, что будет храниться в базе данных, а что - нет, а вторая - содержать только те элементы, которые будут храниться в БД.

Во многих методологиях проектирования ER-модель часто является фактически просто СУБД-независимым описанием логической структуры реляционной БД.

Методология построения ER-модели зависит от изобразительных возможностей, заложенных в язык описания ER-модели, а также от алгоритма перехода от ER-модели к модели базы данных в среде конкретной СУБД, реализованного в CASE-средстве. Кроме того, естественно, методология зависит от особенностей модели БД целевой СУБД. Последнее оказывает влияние не только на методологию, но и на сам язык моделирования предметной области. К сожалению, практически все современные средства ER-моделирования ориентированы на реляционные модели данных, и их использование для других моделей приведет к некорректному результату.

Несмотря на то, что при использовании большинства CASE-средств на основе ER-моделей структура целевой базы данных определяется автоматически, требования к проектировщикам базы данных не только не уменьшаются, но и, напротив, возрастают: специалисты должны не только в совершенстве владеть методологией моделирования предметных областей с использованием языковых средств конкретной системы автоматизации проектирования, но и понимать алгоритм проектирования, заложенный в данном CASE-средстве, а также владеть теорией проектирования баз данных. В противном случае полученное проектное решение может оказаться не просто недостаточно эффективным, но и вообще не соответствовать отображаемой предметной области и содержать ошибки. Алгоритмы проектирования, заложенные в CASE-средствах, как правило, не публикуются. Поэтому, прежде чем приступить к реальному проектированию в среде конкретного CASE-средства, следует поэкспериментировать, посмотреть, как те или иные конструкции ER-модели и их сочетания преобразуются в модель БД.

В разд. 2.2 рассмотрены язык для построения базовой ER-модели и методика его использования, а далее (*см.* разд. 3.3) будет описан алгоритм перехода от этой модели к логической модели реляционной базы данных.

В предлагаемом подходе в ER-модели должны быть отображены все элементы, о которых идет речь в исследуемой предметной области. В процессе проектирования БД выполняется ряд шагов, в том числе и определение состава показателей, хранимых в базе данных. В базу данных могут перено-

ситься не все атрибуты, присутствующие в ER-модели. Так, производные показатели часто не включаются в БД.

Изображение ER-модели, подлежащей преобразованию в модель целевой БД, будет зависеть и от других особенностей алгоритма преобразования инфологической модели в даталогическую. В данном учебнике предложен многоальтернативный алгоритм проектирования, в котором одной и той же конструкции ER-модели может соответствовать несколько проектных решений, которые принимаются проектировщиком на основе анализа многих факторов. К сожалению, далеко не все CASE-системы позволяют проектировщику выбирать проектное решение из нескольких альтернативных.

В предлагаемом автором подходе именно на алгоритм перехода от ER-модели к модели БД возлагаются задачи проектирования структуры базы данных, а ER-модель и другие компоненты инфологической модели должны содержать информацию, достаточную для обоснованного принятия проектного решения. Специалист, строящий ER-модель, в этом случае вообще может ничего не знать о модели и структуре базы данных. Именно такой подход и отвечает сущности инфологического моделирования - описание предметной области безотносительно к используемым в дальнейшем СУБД. Во многих CASE-системах ER-модели ориентированы только на реляционные базы данных; в них каждому объекту ER-модели соответствует таблица базы данных. В этом случае построение ER-модели мало чем отличается от проектирования реляционной базы данных.

Заложенные в систему проектные решения оказывают влияние на методологию построения ER-модели. Так, например, если алгоритм проектирования позволяет не каждому объекту ставить в соответствие таблицу, то для построения ER-модели при решении вопроса о том, какой из сущностей предметной области ставить в соответствие отдельный объект в ER-модели, а когда отображать его в виде характеристического свойства, можно порекомендовать выделять отдельный объект, не содержащий никаких атрибутов, кроме идентификатора, если он участвует в нескольких связях. В этом случае ER-модель получается менее громоздкой, так как содержит меньше элементов. При использовании CASE-средств такое решение еще более удобно, чем при ручном проектировании, поскольку при этом происходит автоматическая миграция соответствующих атрибутов. Придерживаться же данной выше рекомендации при построении ER-модели с использованием тех CASE-систем, в которых каждому объекту ER-модели соответствует таблица в реляционной базе данных, нельзя. Поэтому необходимо уже на стадии ER-моделирования решить, каким сущностям ставить в соответствие таблицы, а каким - нет, и в качестве объектов ER-модели изображать только те сущности, которым будут соответствовать отдельные таблицы.

Рассмотрим еще некоторые примеры влияния языка и алгоритма проектирования на построение ER-модели. В базовой ER-модели введено понятие *условное свойство,* которого нет в других методологиях ER-моделирования. Оно означает, что свойство может присутствовать не у всех объектов данного

класса. При изображении таких ситуаций в методологиях типа IDEF1X, где нет соответствующего понятия, возможно несколько вариантов:

1) условное свойство изображать как обычный атрибут;

2) объект, обладающий условным свойством, изобразить как обобщенный объект (при этом условное свойство будет принадлежать видовому объекту);

3) выделить «обладание свойством» в отдельный объект и при установлении связи этого вновь введенного объекта с исходным объектом соответствующим образом определить характеристики этой связи.

В тех CASE-системах, которые позволяют выбирать проектное решение при преобразовании ER-модели в модель целевой СУБД (как, например, в PowerDesigner), лучше использовать вариант 2. В тех системах (как, например, Design/IDEF), в которых каждому видовому объекту ставится отдельная таблица, следует сначала принять решение, как вы хотите хранить информацию в БД, а только затем выбирать вариант отображения предметной области в ER-модели.

Большинство CASE-систем содержат изобразительные средства для отображения *обобщенных объектов,* хотя как используемые для этих целей условные обозначения, так и алгоритм отображения этих конструкций в даталогическую модель отличаются от системы к системе. Так, например, в методологии IDEF1X можно проводить классификацию объектов по нескольким независимым признакам. В CASE Oracle это сделать невозможно (можно изобразить только строго иерархическую, а не фасетную классификацию). Это, безусловно, скажется на подходе к моделированию предметной области: при невозможности отобразить многоаспектную классификацию в большинстве случаев придется подклассы изображать как самостоятельные объекты.

Понятие множественного свойства также отсутствует в большинстве CASE-систем. Для изображения каждого множественного свойства приходится использовать отдельный объект, зависящий по идентификации от основного объекта, обладающего этим свойством. Атрибут, соответствующий множественному свойству, должен быть отмечен как ключевой.

Многие CASE-системы (Design/IDEF, ERWin, CASE Oracle) позволяют изображать связь M:M на начальных этапах построения ER-модели, но перед выполнением этапа генерации схемы БД эта связь должна быть преобразована путем введения дополнительной связующей сущности и связей 1 :M с ней. В Design/IDEF и ERWin при изображении связи M:M нельзя указать класс принадлежности объектов в этой связи. Если класс принадлежности является необязательным хотя бы с одной из сторон связи, то приходится либо применять искусственные приемы, чтобы адекватно отразить характер связи, либо терять часть информации о предметной области. В качестве такого искусственного приема для систем, базирующихся на методологии IDEF1X, можно предложить следующий: класс объектов, класс принадлежности которого в данной связи «необязательный», делится только на один подкласс (естественно, что в этом случае тип подкласса будет incomplete - неполный), и уже этот видовой объект используется в связи.

В базовой модели введено понятие «имя объекта» - то, что называет объект. Имя может быть как уникальным, так и неуникальным. Это фиксируется в модели. Во всех остальных проанализированных методологиях выделяется атрибут или совокупность атрибутов, соответствующих первичному ключу. Если у объекта есть несколько даже уникальных имен, то только одно из них должно быть выбрано в качестве идентификатора уже на стадии концептуального моделирования. Другими словами, проблема выбора ключа реляционной таблицы переносится на стадию инфологического проектирования, что не совсем оправданно.

Понятия «ключ» и «имя объекта» - это не всегда одно и то же. В качестве идентификатора может быть выбрана совокупность атрибутов, не обязательно принадлежащих имени (например, добавление даты и места рождения при идентификации личности в поисковых системах).

ER-модели служат не только непосредственной цели проектирования базы данных. Они выполняют и другие, более общие функции, такие, как понятное, полное, формализованное описание предметной области и использование его для обсуждения как с заказчиками/ пользователями проектируемой системы, так и с разработчиками разной специализации. Поэтому можно рекомендовать при проектировании ИС с использованием таких CASE-систем, которые не позволяют определять, какие из объектов/свойств будут храниться в базе данных, строить две модели:

1) модель предметной области, содержащую все объекты, как подлежащие, так и не подлежащие хранению в БД, и использующую изображение множественных связей;

2) концептуальную модель БД, которая затем автоматически будет преобразована в модель целевой БД.

Недостаточные изобразительные возможности языковых средств, используемых при изображении ER-моделей, с одной стороны, приводят к обеднению этих моделей, а с другой, как это ни парадоксально звучит, - усложняют процесс построения ER-модели и делают эти модели более громоздкими.

2.5. Использование Design/IDEF для проектирования баз данных

2.5.1. Построение ER-модели при использовании Design/IDEF

Общая характеристика

Design/IDEF является комплексной системой автоматизации проектирования ИС. Она объединяет в себе несколько методологий, каждая из которых предназначена для построения моделей определенного типа. Для построения ER-модели используется методология IDEF1X.

Нотация языка ER-моделирования, алгоритм проектирования целевой реляционной модели и, как следствие, подход к моделированию существенно отличаются от базовой модели. Изобразительные средства нотации языка

ER-моделирования в Design/IDEF более бедные, чем в базовой модели. Процесс построения ER-модели в Design/ IDEF практически сводится к описанию реляционной модели данных другими изобразительными средствами (каждому объекту ER-модели в Design/IDEF соответствует таблица реляционной базы данных). В связи с этим рекомендуется сначала изучить главу 3, а потом, используя эти знания, строить ER-модель в нотации Design/IDEF.

Укрупненная схема работы с Design/IDEF при построении ER-модели показана на рис. 2.39.

После загрузки Design/IDEF и выбора команды **File/New** экран имеет вид, изображенный на рис. 2.40. В окне **Methodology** будет высвечиваться название той методологии, с которой работали в предыдущем сеансе. Если это не методология IDEF1X, то следует нажать на кнопку в правой стороне этого окна, из ниспадающего списка выбрать требуемую методологию (рис. 2.41).

Начальный экран проектирования после выбора методологии IDEF1X изображен на рис. 2.42.

После этого можно приступить к построению ER-модели, а можно сначала выбрать целевую СУБД - ту СУБД, для которой ведется проектирование базы данных. Для этого следует выбрать команду **Edit/ Set Options.** В появившемся окне **IDEF Options** следует активизировать переключатель IDEF1X (рис. 2.43) и в окне **Target Datadase** выбрать нужную СУБД из ниспадающего списка поддерживаемых системой СУБД. Выбор целевой СУБД окажет влияние на список допустимых типов данных при описании атрибутов. На методологию построения модели этот выбор влияния не оказывает. Выбор целевой СУБД можно провести на любом этапе проектирования. При этом типы данных будут автоматически преобразованы в соответствующие им типы в новой целевой СУБД.

Описание сущности

Построение новой ER-модели можно начать только с создания новой сущности. Для того чтобы построить новую сущность, надо щелкнуть по третьей кнопке сверху (прямоугольник, разделенный на два сектора) на панели инструментального меню, изображенного слева на экране (или выполнить команду **Create/Entity,** или нажать комбинацию клавиш [Ctrl] +[E]), и, позиционировав курсор с «прикрепленным» к нему квадратом в нужном месте экрана, нажать левую кнопку мыши. На экране появится окно определения сущности **(Define Entity),** изображенное на рис. 2.44. Система автоматически присваивает каждой создаваемой сущности уникальный идентификатор **(Entity ID).** В поле Name следует указать имя сущности. Кроме основного имени можно задать еще и **Aliases** (псевдоним) - имя, используемое в качестве синонима

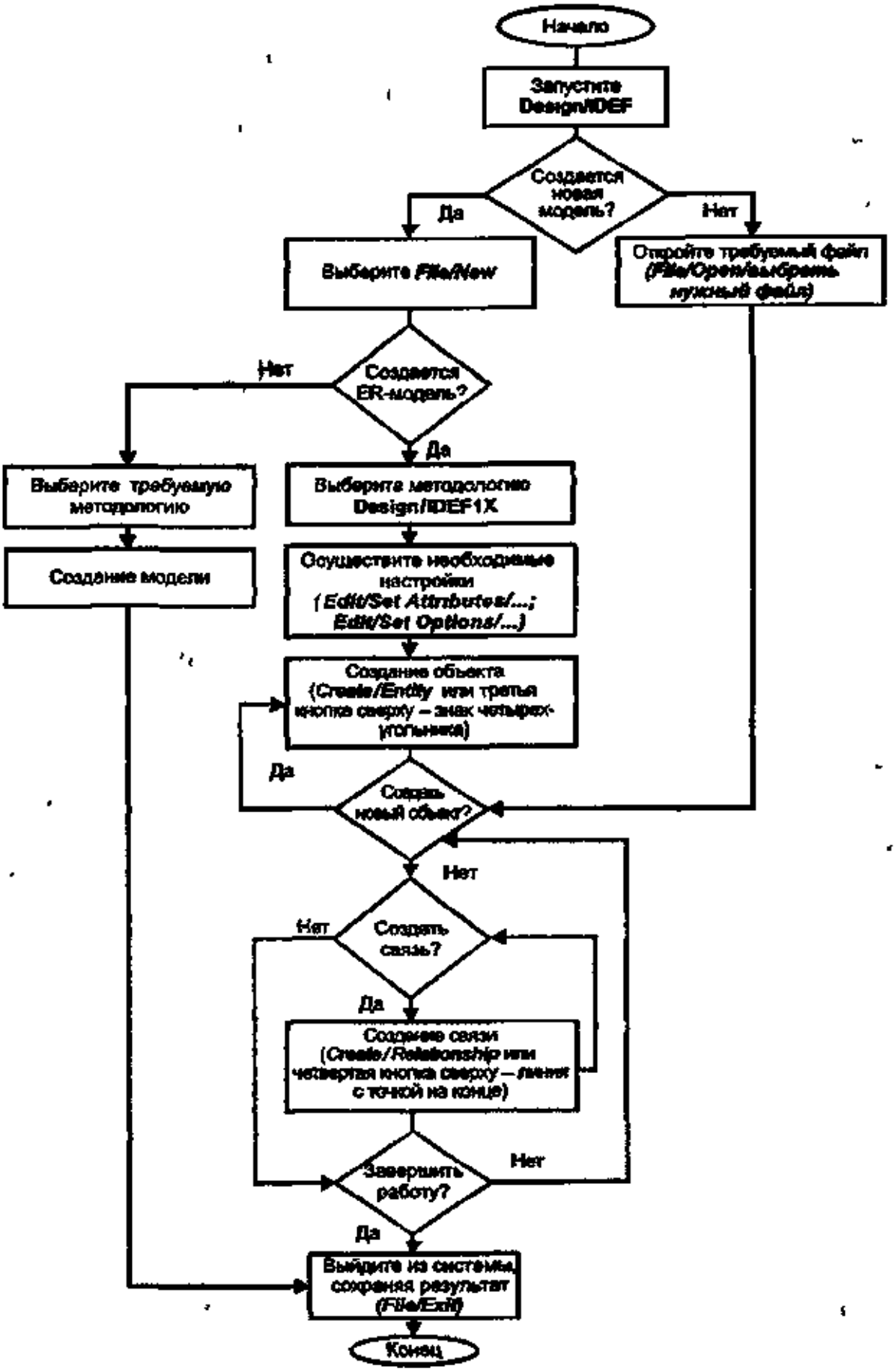


Рис. 2.39. Укрупненная схема создания модели Design/IDEF

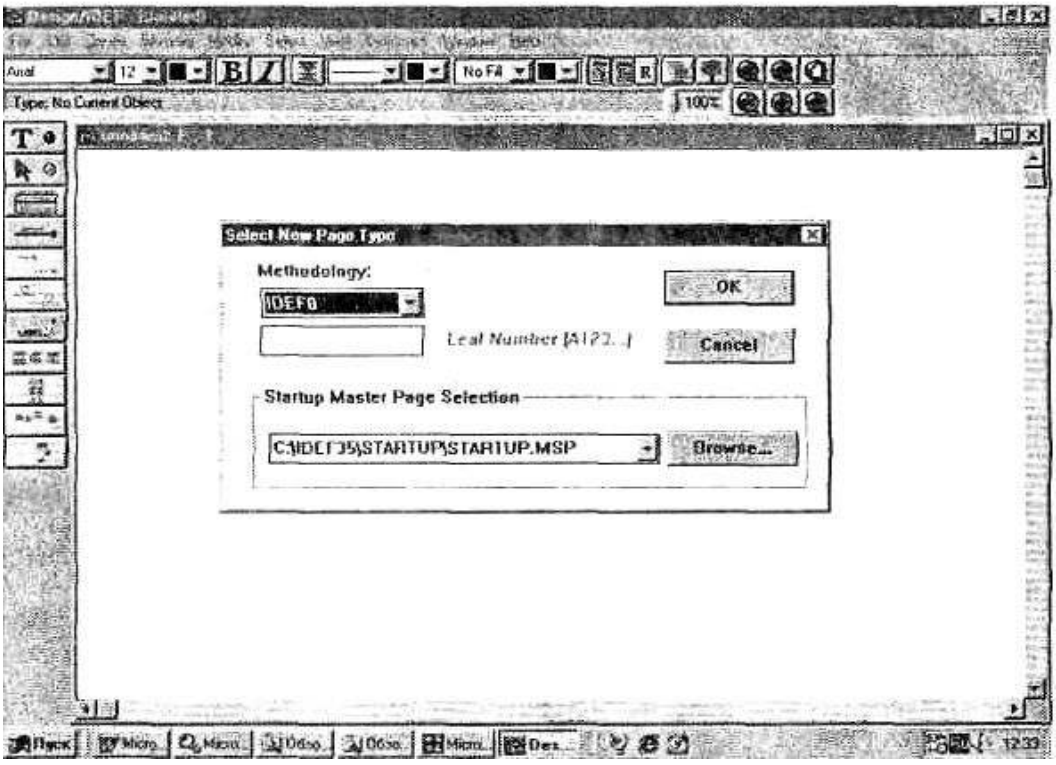


Рис. 2.40. Вид экрана после загрузки системы Design/IDEF
и выбора команды File/New

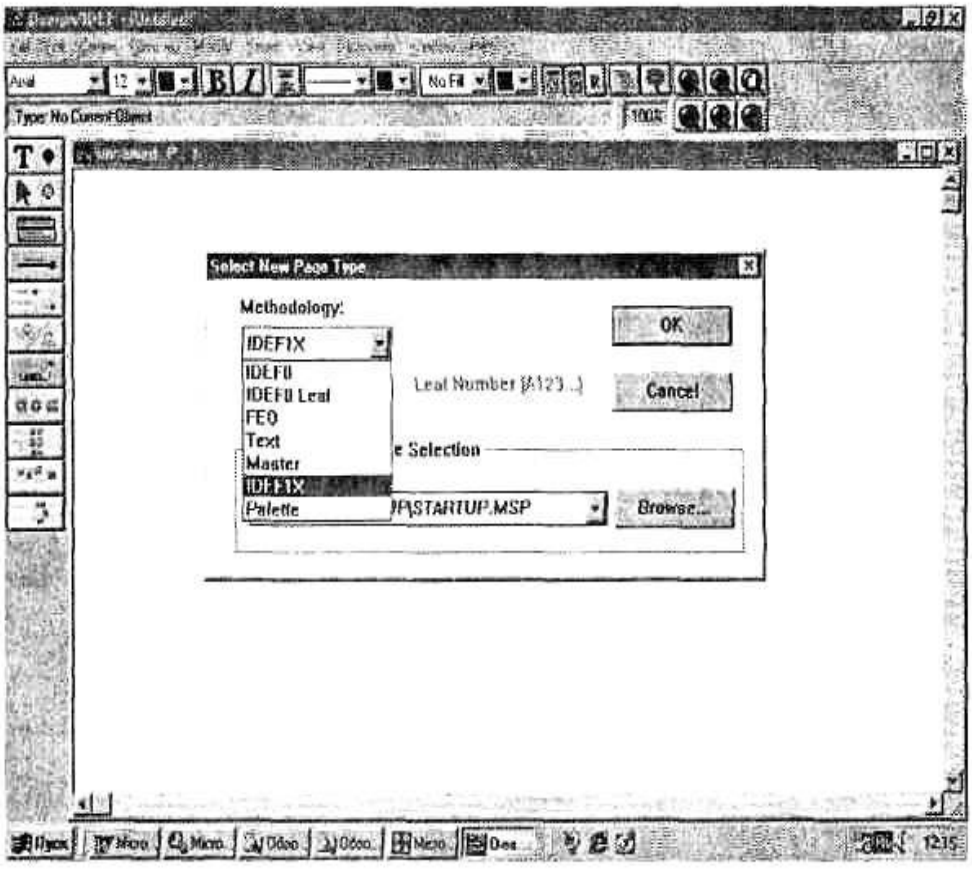


Рис. 2.41. Вид окна при выборе методологии

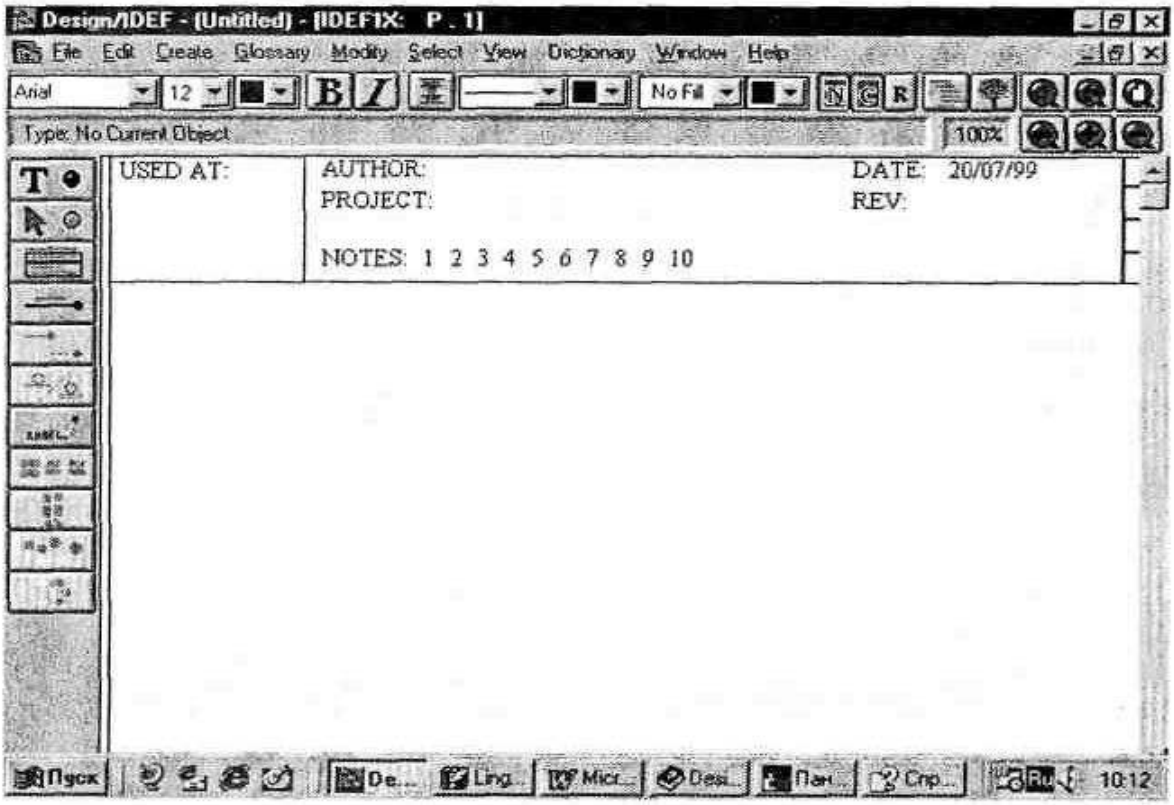


Рис. 2.42. Начальный вид окна проектирования
после выбора методологии IDEF1X

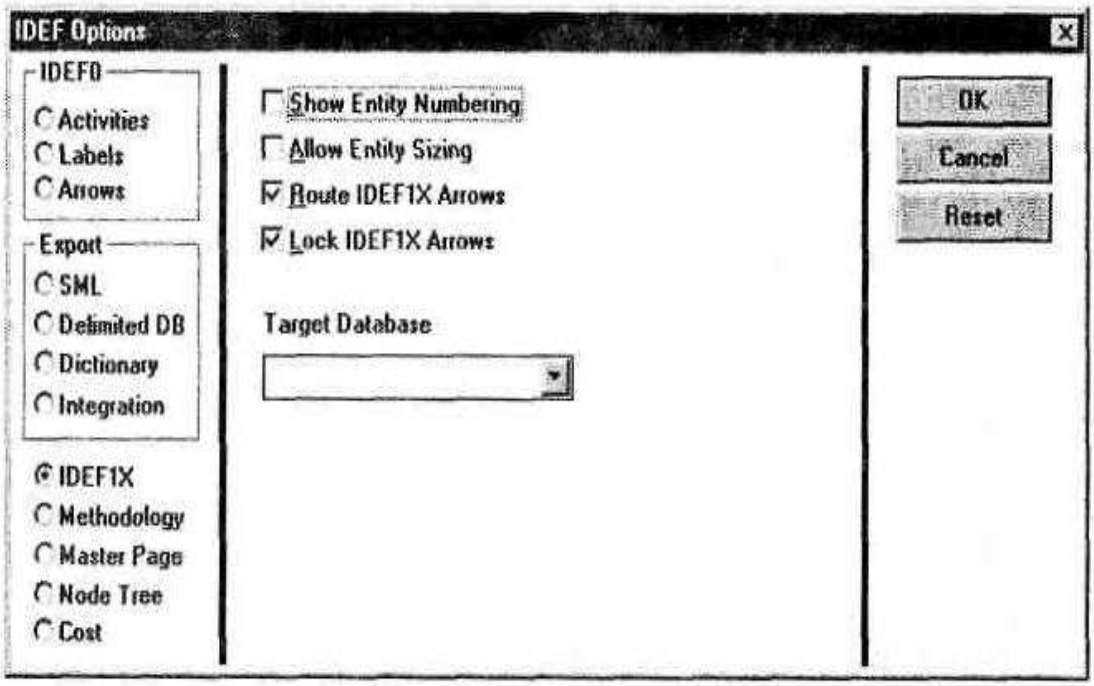


Рис. 2.43. Вид окна задания опций

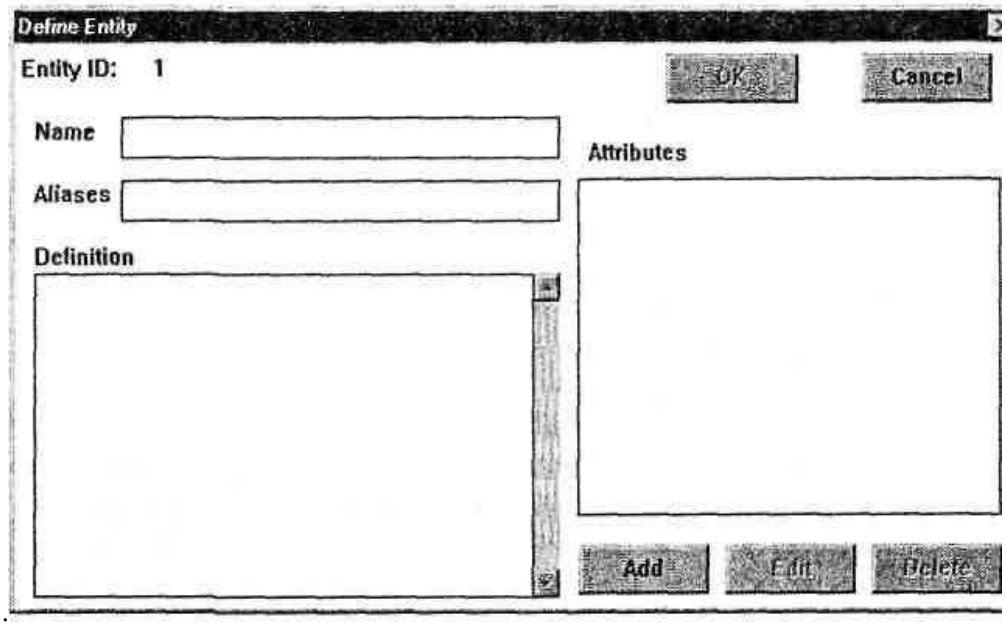


Рис. 2.44. Вид окна определения сущности

В окне **Definition** можно дать описание сущности. Такие описания важны для документирования модели, уточнения терминологии, используемой в проекте, но на проектировании структуры базы данных они не скажутся.

После этого следует перейти к описанию атрибутов сущности и щелкнуть по кнопке **Add** в окне определения сущности. При этом появится окно определения атрибута **(Define Attribute),** изображенное на рис. 2.45.

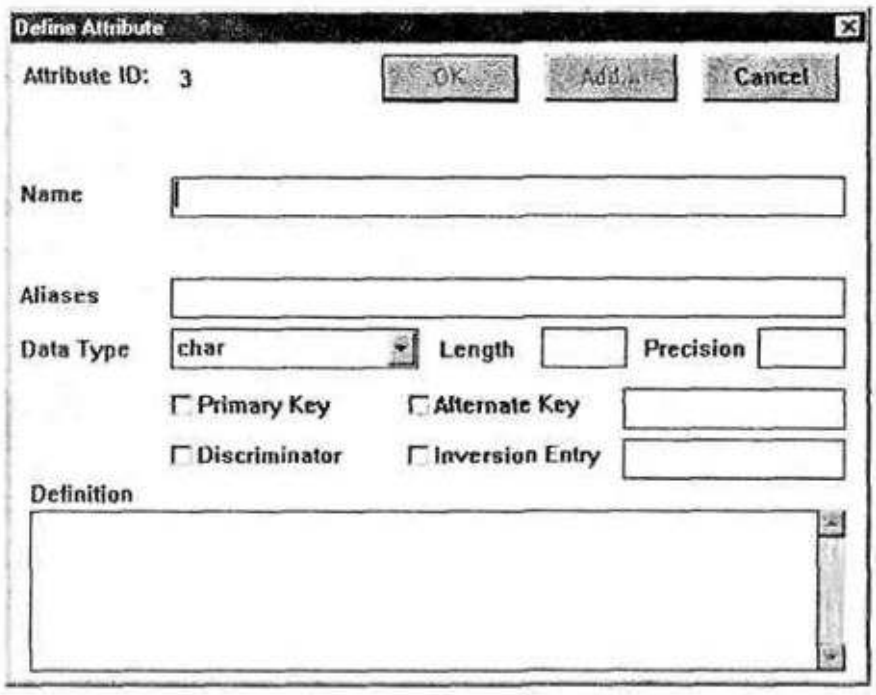


Рис. 2.45. Вид окна определения атрибута

В поле **Name** следует записать наименование атрибута, в поле **Data Type** - выбрать тип, присущий данному атрибуту. Список доступных типов полей зависит от выбранной целевой СУБД. Как известно, некоторые типы полей

имеют фиксированную длину, и для них нет нужды указывать длину (Length), для других - должна указываться длина; для числовых полей с дробной частью нужно указывать точность (Precision).

В реляционной теории есть понятия «ключ» и «вероятный ключ». Эти понятия характеризируют не предметную область, а именно таблицу реляционной базы данных. При преобразовании ER-модели в схему реляционной базы данных в Design/IDEF каждой сущности ставится в соответствие таблица реляционной базы данных, и при описании сущности требуется определить, какой атрибут (или несколько атрибутов) выбран в качестве первичного ключа. Эти атрибуты должны быть помечены галочкой в переключателе **Primary Key.**

Для нескольких атрибутов можно указать, что он является РК. На самом деле это означает, что каждый из таким образом обозначенных атрибутов является элементом составного ключа, а не самостоятельным ключом, что с точки зрения реляционной теории принципиально важно различать.

Если есть несколько альтернативных (вероятных) ключей, то нужно все остальные описать как альтернативные. Вероятные ключи помечаются галочкой в переключателе **Alternate Key.** Так как вероятных ключей может быть несколько, то в поле справа от свойства **Alternate Key** необходимо поставить порядковый номер. Если альтернативный ключ является составным, то все составляющие его атрибуты следует пометить одним и тем же номером в поле рядом с переключателем **Alternate Key**.

При проектировании баз данных учитывается множество разных факторов, и в том числе характер запросов. Если по какому-либо полю часто осуществляется выборочный поиск, то по такому полю обычно проводят индексирование. Такие проиндексированные поля иногда называют «инверсным входом». При описании сущности именно такие атрибуты следует пометить галочкой в переключателе **Inversion Entry.** Инверсных входов, так же как и вероятных ключей, может быть несколько. Поэтому в поле справа от свойства **Inversion Entry** необходимо поставить порядковый номер.

Как известно (*см.* разд. 2.2), объекты бывают не только простыми, но и обобщенными. То свойство, по которому классы делятся на подклассы, в Design/IDEF называется «дискриминатором» (Discriminator). При описании соответствующего свойства следует это отметить галочкой в переключателе **Discriminator**.

Назначение свойств **Name** и **Aliases** в окне определения атрибута аналогично соответствующим характеристикам, задаваемым при описании самого объекта в окне определения сущности.

После описания очередного атрибута следует опять щелкнуть по кнопке **Add** и задать описание следующего атрибута. После того как определены все атрибуты описываемого объекта, щелкните по кнопке **ОК**.

На рис. 2.46 приведен фрагмент описания сущности СОТРУДНИК. Если на этом завершить описание атрибутов и щелкнуть по кнопке **ОК**, то получим графическое представление описанного объекта (рис. 2.47).

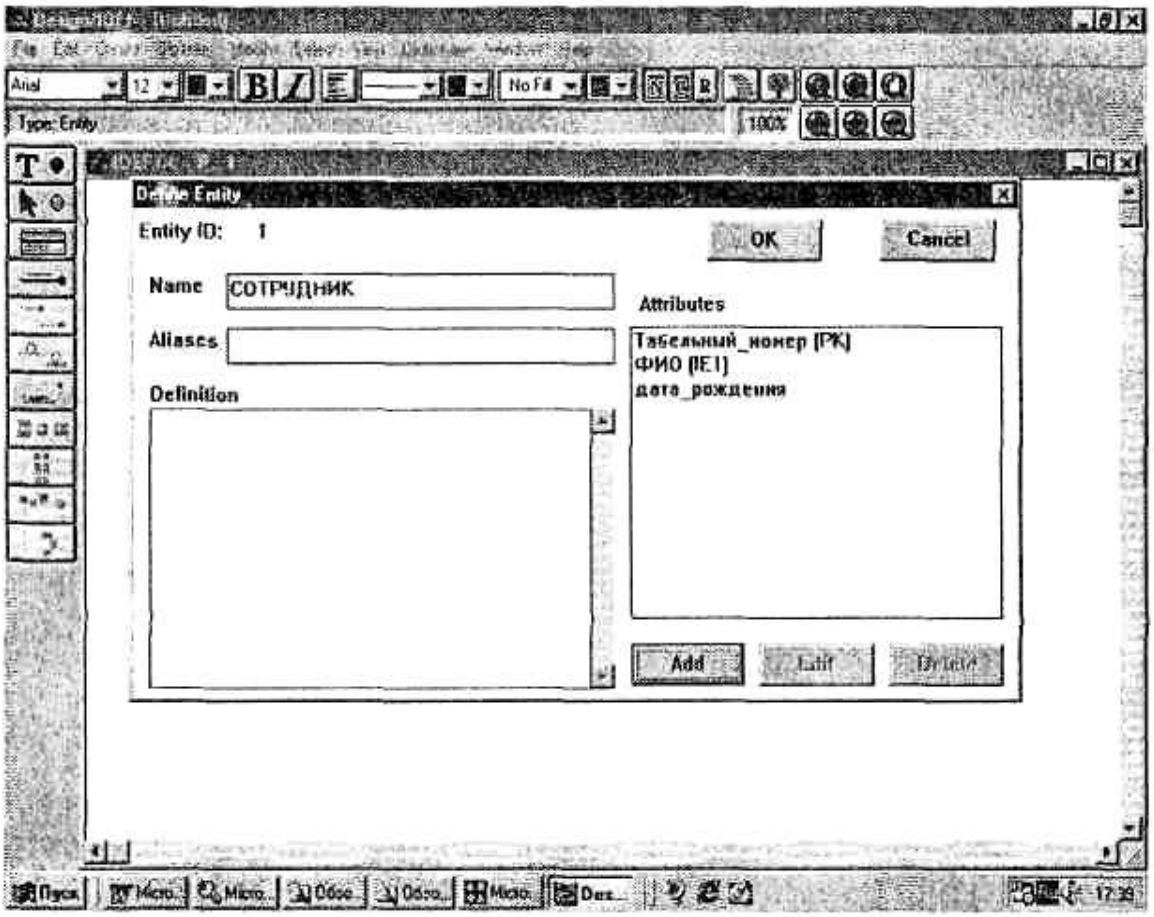


Рис. 2.46. Экран описания объекта (пример)

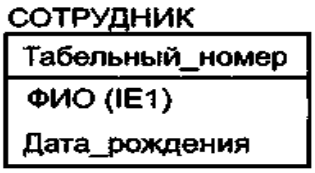


Рис. 2.47. Пример изображения объекта (вариант 1)

В общем виде графическое представление сущности отражено на рис. 2.48. Для обозначения сущности используют прямоугольник, разделенный на два сектора; в верхнем секторе записываются ключевые атрибуты, в нижнем - все остальные. Рядом с атрибутами, являющимися альтернативными ключами, в скобках ставятся буквы (АК).

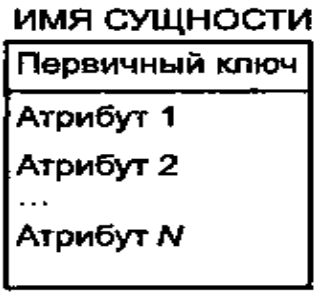


Рис. 2.48. IDEF1X ERD. Обозначение объекта

Если вы уверены, что ФИО вместе с Датой_рождения является уникальной совокупностью атрибутов и их следует описать как вероятный ключ, то графическое представление подобного объекта будет таким, как это показано на рис. 2.49. Несмотря на то, что в данном примере ФИО вошло в альтернативный ключ, его следует оставить и как инверсный вход, поскольку по ФИО часто осуществляется поиск.

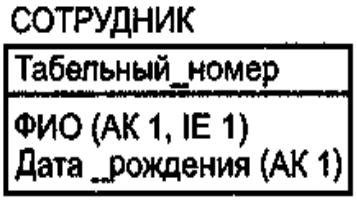


Рис. 2.49. Пример изображения объекта (вариант 2)

Предыдущий пример следует рассматривать именно как технический пример создания составного альтернативного ключа и не более того. В реальной жизни вряд ли целесообразно создавать такой альтернативный ключ. Нужно понимать, что не все потенциальные альтернативные ключи реляционной таблицы следует обозначать в ER-модели. Это надо делать только тогда, когда есть необходимость контролировать уникальность альтернативного ключа.

Альтернативных ключей у одной сущности может быть несколько (рис. 2.50). Обратите внимание на разницу в изображении сущностей на рис. 2.49 и 2.50. В первом случае сущность имеет один составной альтернативный ключ, во втором - два простых.

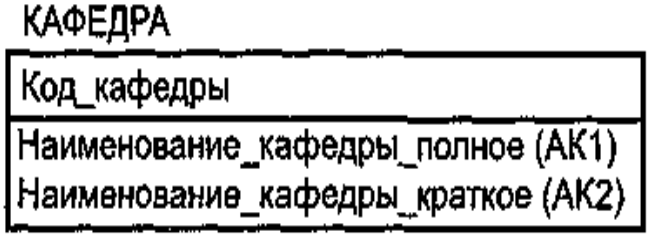


Рис. 2.50. Изображение сущности КАФЕДРА

Описание связи

После того как были определены хотя бы две сущности, можно задавать связи между ними. Для определения связи можно щелкнуть по четвертой кнопке сверху (линия с точкой на конце) на панели инструментов, изображенной слева на экране (или выполнить команду **Create/Relationship,** или нажать комбинацию клавиш [Ctrl]+[R]). После этого курсор с «прикрепленной» к нему линией со стрелочкой надо позиционировать на значке того объекта, от которого должна начинаться связь, и, не отпуская левую клавишу

мыши, протянуть связь до того объекта, на котором она должна оканчиваться.

После этого появится окно определения связи **Define Relationship** (рис. 2.51). В первой строке этого окна **(Entity)** указывается имя исходного объекта, от которого идет связь; в третьей строке с таким же названием **(Entity)** - имя объекта, к которому направлена связь. В окне **Relationship** обязательно должно быть указано имя связи в прямом направлении (в нашем примере это связь от объекта ГРУППА к объекту СТУДЕНТ и назвать ее можно, например, «включает» или «состоит»). В окне **Inverse** можно (но не обязательно) задать имя связи в обратном направлении (для нашего примера это может быть имя «учится»). Переключатель **Separate Lines** становится активным только в том случае, если имя обратной связи задано. Этот переключатель определяет только то, как на экране располагаются надписи прямой и обратной связи.

Для каждой связи должен быть определен тип связи **(Relationship Type**) и кардинальное число **(Relationship Cardinality).**

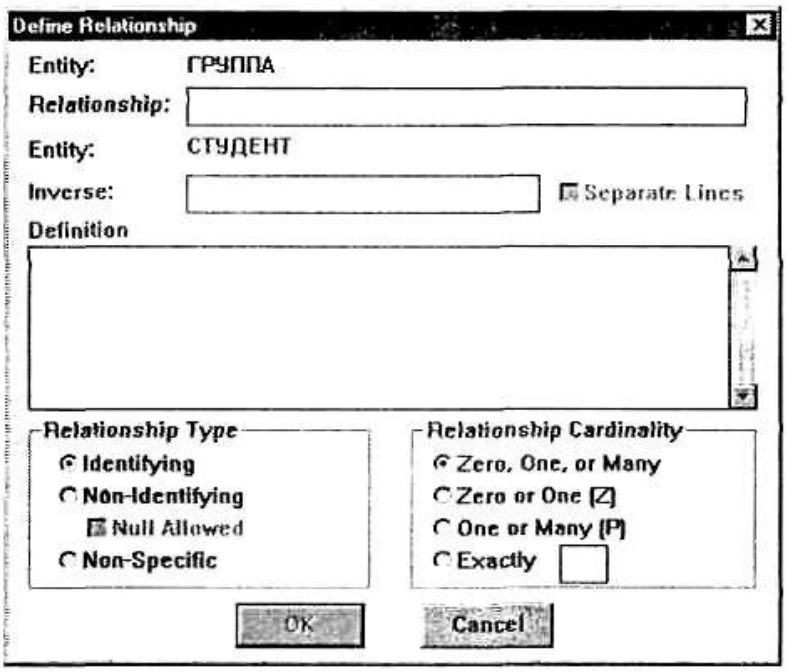


Рис. 2.51. Вид окна описания связи между объектами

В Design/IDEF различают следующие типы связи:

- Identifying (идентифицирующая);
- Non-Identifying (неидентифицирующая);
- Non-Specific (неспецифическая).

Идентифицирующая и неидентифицирующая связи представляют собой связи 1:M. Первая из них используется, если связь направлена к не зависимой по идентификации сущности (*см.* разд. 2.1.1), вторая - к зависимой. Неспецифическая связь представляет собой связь M:M между объектами. Она названа так потому, что реляционная модель не поддерживает связи M:M. В

Design/IDEF неспецифические связи могут быть использованы на начальных этапах моделирования. Перед генерацией схемы базы данных ER-модель нужно преобразовать и заменить неспецифические связи специфическими путем введения дополнительных связующих объектов. Если этого не сделать, то схема БД будет не адекватна предметной области.

На рис. 2.52 изображены все возможные виды связи в Design/IDEF. Идентифицирующая (сплошная линия) и неидентифицирующая (пунктирная линия) связи отличаются не только видом линии, но и тем, куда мигрирует ключ исходной сущности: в первом случае он становится частью ключа «целевой» сущности, а во втором - неключевым атрибутом (но в том и в другом случае он является внешним ключом - FK).

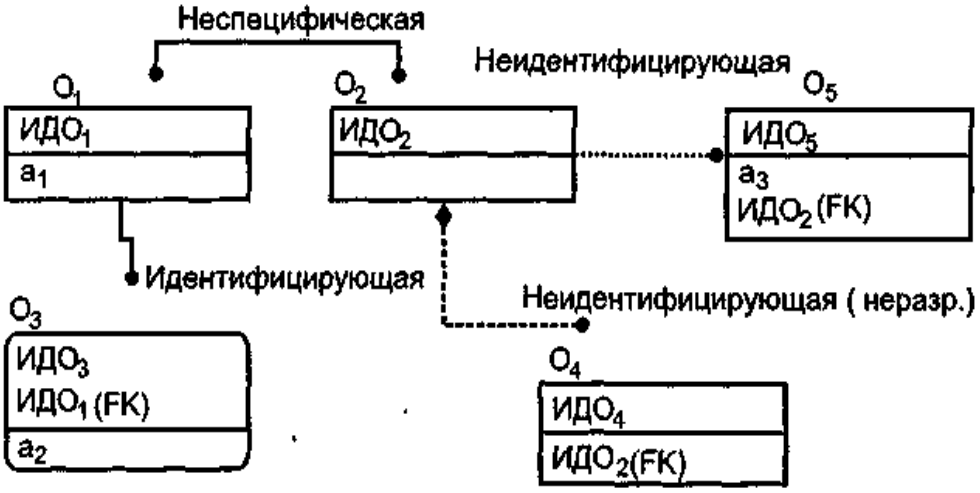


Рис. 2.52. Виды связи в Desing/IDEF1X

На рис. 2.53 приведены примеры использования идентифицирующей и неидентифицирующей связи. Выбор варианта будет определяться тем, как обозначаются участки в конкретной предметной области. Если выбрана неидентифицирующя связь, то возможно задание дополнительной характеристики этой связи - разрешаются ли «пустые», неопределенные значения внешнего ключа **(Null Allowed).** На рис. 2.53 изображены ситуации и с разрешенными (рис. 2.53, б), и с запрещенными (рис. 2.53, *а)* Null-значениями. Но это сделано для того, чтобы показать, как графически различается изображение в каждой из этих ситуаций. Выбор Null Allowed означает, что возможно наличие участков, не приписанных ни к какому цеху, что в реальных предметных областях скорее всего недопустимо.

Если к сущности идет идентифицирующая связь, то сущность называется зависимой по идентификации. Такая сущность, как мы видим, изображается прямоугольником с закругленными углами. Ключ зависимой по идентификации сущности чаще всего бывает составным, одним из элементов которого является ключ исходного объекта (рис. 2.53, *в*). Идентифицирующую связь нельзя провести, если у исходного объекта не задан ключ. Знак зависи-

мой по идентификации сущности нельзя внести в схему непосредственно; углы закругляются автоматически, при объявлении соответствующей связи.

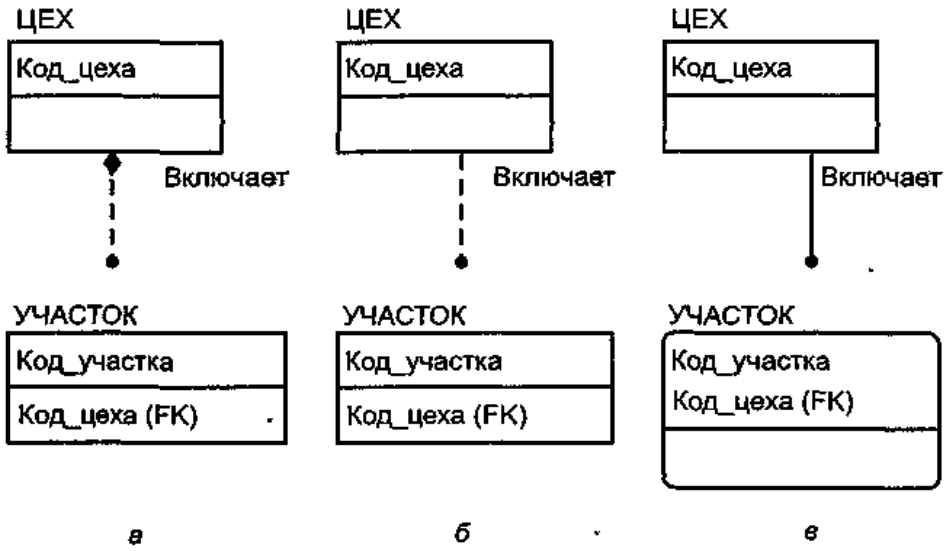


Рис. 2.53. Примеры изображения связи между объектами:
а - неидентифицирующая (не разрешено «пустое» значение);
б - неидентифицирующая («пустое» значение разрешено);
в - идентифицирующая

Кроме указания типа связи для каждой связи задается кардинальное число (*см.* рис.2.51), указывающее количественные характеристики связи. В IDEF1X могут быть выражены следующие мощности связей:

- каждый экземпляр сущности-родителя может иметь нуль, один или более одного связанного с ним экземпляра сущности-потомка (*Zero, One or Many);*
- каждый экземпляр сущности-родителя должен иметь не более одного связанного с ним экземпляра сущности-потомка (*Zero or One [Z]);*
- каждый экземпляр сущности-родителя должен иметь не менее одного связанного с ним экземпляра сущности-потомка (*One or Many [P])*
- каждый экземпляр сущности-родителя связан с некоторым фиксированным числом экземпляров сущности-потомка (*Exactly).*

В последнем из рассмотренных случаев кроме того, что в него устанавливается переключатель, в окне следует указать число. Например, если набираются группы для обучения в компьютерных классах и занятия не начинаются, пока не наберется 10 человек, но и большее число учащихся недопустимо, то следует выбрать вариант *Exactly* и поставить число 10.

На рис. 2.54 приведены графические представления всех допустимых в IDEF1X типов мощности связи, на рис. 2.55 - примеры использования разной кардинальности связи.

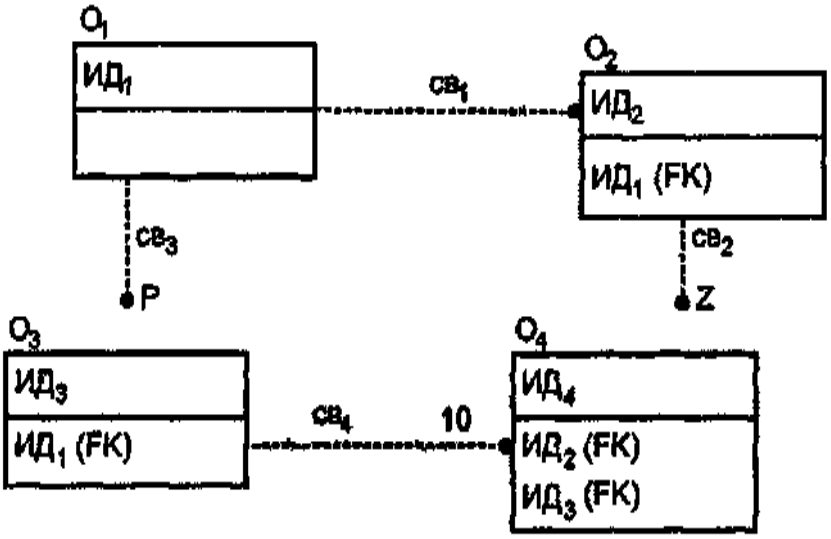


Рис. 2.54. IDEF1X. Виды мощности связи:
$св_1$ - Zero, One or Many; $св_2$ - Zero or One [Z];
$св_3$ - One or Many [P]; $св_4$ – Exactly

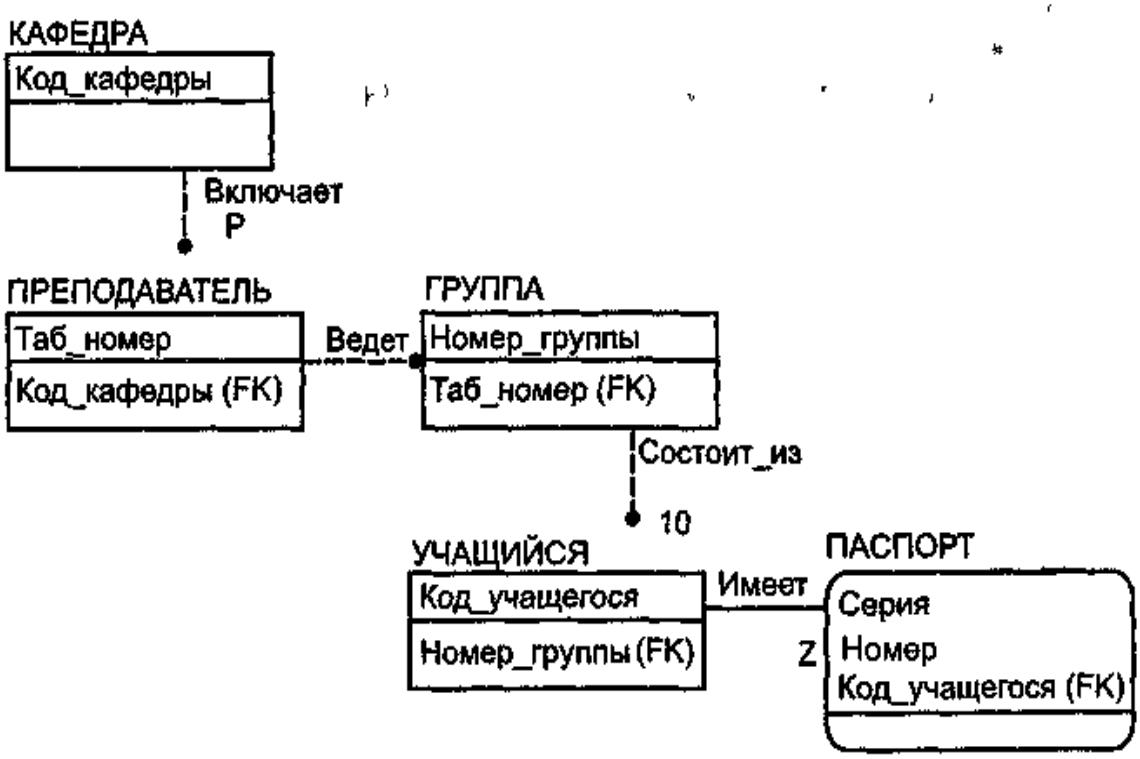


Рис. 2.55. Фрагмент ER-модели
(пример использования разной кардинальности связи)

Описание обобщенного объекта

В Design/IDEF нет специального графического значка для обозначения обобщенного объекта. Но возможность отображать обобщенные объекты есть. Для этого следует изобразить сначала родовую сущность. Для нее нуж-

но указать все идентификаторы изображаемого объекта, общие атрибуты, присущие всем объектам класса, а также тот атрибут, по которому класс разбивается на подклассы (такой атрибут следует обозначить как дискриминатор). На рис. 2.56 показан вид экрана описания родовой сущности, а на рис. 2.57 - соответствующее ему графическое представление. Как видно из рис. 2.57, дискриминатор на схеме изображается окружностью с прилегающими к ней двумя параллельными линиями. При объявлении дискриминатора всегда первоначально появляется такой знак. Он означает полное разбиение класса на подклассы (т.е. каждый элемент класса обязательно относится к одному из выделенных подклассов). Если это не так, то нужно выделить значок дискриминатора, после чего воспользоваться переключателем «Полный-неполный дискриминатор» (шестая кнопка сверху в инструментальном меню) или позицией меню **Create/Toggle Discriminator.**

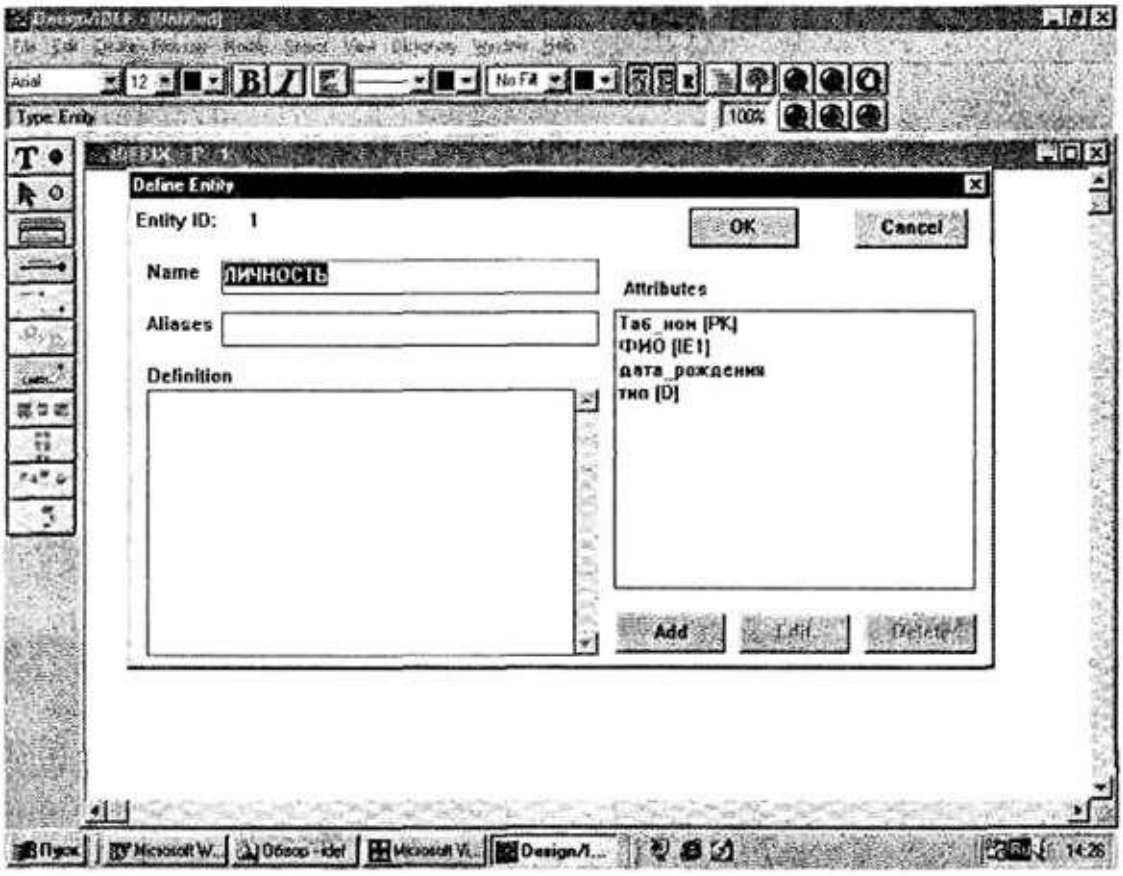


Рис. 2.56. Вид экрана описания родового объекта



Рис. 2.57. Графическое представление родового объекта

Далее необходимо изобразить сущности, соответствующие видовым объектам. При этом ключевые поля описывать не надо. В качестве атрибутов этих сущностей следует описывать те атрибуты, которые присущи именно этому подклассу (рис. 2.58). Затем нужно провести связи от значка дискриминатора к значкам видовых объектов, после чего схема примет вид, показанный на рис. 2.59. При этом ключи родового объекта автоматически мигрируют в видовые объекты, а значки видовых объектов станут иметь закругленные углы.

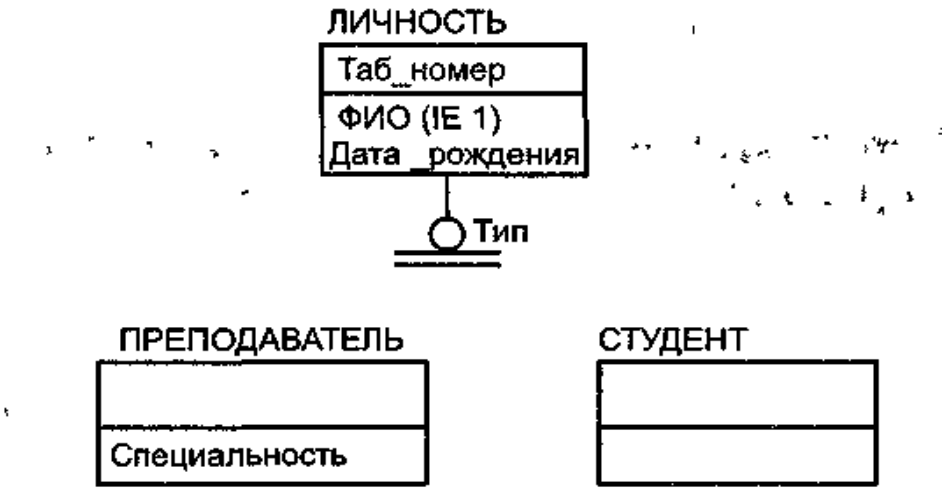


Рис. 2.58. Графическое представление видовых сущностей (промежуточный этап)

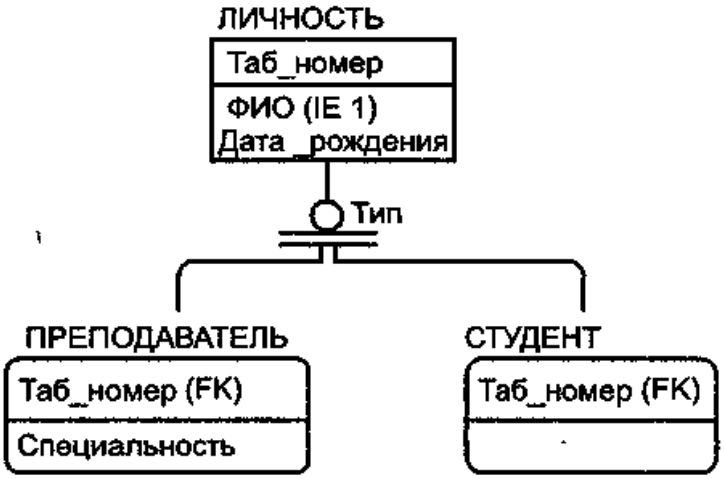


Рис. 2.59. IDEF1X ERD. Пример изображения обобщенного объекта

Изображение обобщенного объекта может выглядеть по-разному (рис. 2.60). Если дискриминатор имеет вид окружности (рис. 2.60, *а*), подчеркнутой двойной линией, это означает, что подклассы в совокупности составляют полный класс. Если дискриминатор изображен в виде окружности (рис. 2.60, *б*), подчеркнутой одинарной линией, - значит, подклассы в совокупности не составляют полный класс. На рис. 2.60, *в* представлен вариант, показываю-

щий возможность классификации сущностей по разным несоподчиненным признакам.

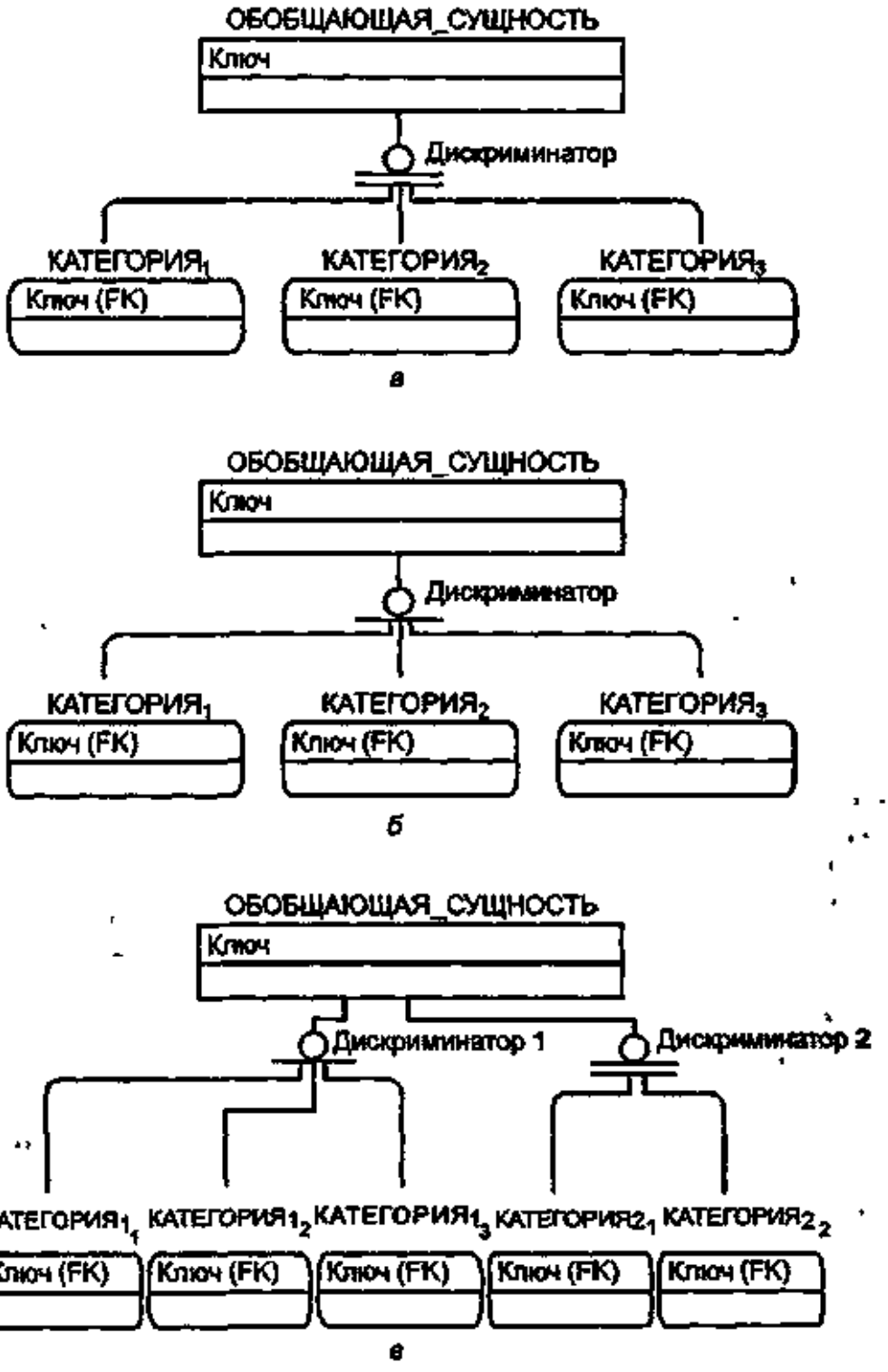


Рис. 2.60. Варианты изображения обобщенного объекта:
а - полный класс; *б* - неполный класс; *в* - классификация по несоподчиненным признакам

На рис. 2.61 изображен пример обобщенного объекта, классифицированного по нескольким признакам.

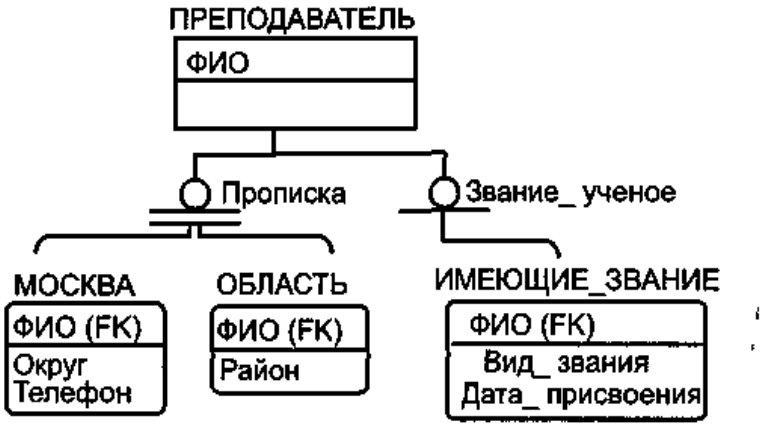


Рис. 2.61. Пример изображения обобщенного объекта, классифицированного по нескольким признакам

2.5.2. Методология построения ER-модели при использовании Design/IDEF

Построение ER-модели является центральным моментом проектирования автоматизированных информационных систем при использовании соответствующих технологий. Этот этап проектирования требует высокой квалификации исполнителей и оказывает существенное влияние на качество всех получаемых в результате проектных решений. Важность качественного ER-моделирования увеличивается еще и в связи с тем, что даже теоретически невозможно создать систему, позволяющую автоматически проверять правильность ER-модели в аспекте ее адекватности отображаемой предметной области.

В разд. 2.3.4 были изложены в общем виде рекомендации по построению ER-модели в зависимости от изобразительных средств и алгоритмов перехода от ER-модели к логической модели реляционной базы данных. Здесь эти рекомендации конкретизированы для системы Design/IDEF.

В Design/IDEF при преобразовании ER-модели в описание целевой базы данных каждому объекту ставится в соответствие таблица реляционной базы данных. Поэтому в отличие от базовой модели, в которой мы отображали все объекты предметной области, в Design/ IDEF нужно сначала определить не просто то, что будет храниться в базе данных, а с уточнением, что будет храниться в отдельной таблице, и с учетом этого строить модель. В качестве объектов ER-модели следует изображать только те сущности, которым будут соответствовать отдельные таблицы. Другой путь - сначала построить предварительную ER-модель, а потом ее преобразовать к варианту, который будет служить исходным для генерации схемы базы данных. Все атрибуты или сущности, соответствующие вычисляемым значениям, которые проектировщик не хочет хранить в базе данных, должны быть устранены из конечной ER-модели.

Как мы видели, в Design/IDEF набор выразительных средств, предоставляемых для построения ER-модели, невелик.

Основным элементом модели является *сущность* (Entity). Сущность имеет имя. Для сущностей описываются их атрибуты. Атрибут может играть роль первичного ключа (Primary Key), альтернативного ключа (Alternate Key), дискриминатора (Discriminator) и инверсного входа (Inversion Entry) либо не играть ни одну из них. Как видно, во-первых, эти характеристики атрибутов отражают совершенно разные аспекты как предметной области, так и организации данных; во-вторых, некоторые из этих показателей жестко привязаны к реляционной модели (понятия первичного и альтернативного ключа); в-третьих, выбор большинства характеристик должен являться результатом проектных решений, обычно выполняемых на основе анализа разнообразных факторов; в-четвертых, ряд характеристик, которые можно отобразить в базовой ER-модели, с которой будет идти дальнейшее сравнение, в методологии IDEF1X в явном виде отобразить нельзя.

В Design/IDEF при преобразовании ER-модели в описание целевой базы данных каждому объекту ставится в соответствие таблица реляционной базы данных.

В Design/IDEF нет изобразительных средств для обозначения множественного свойства, составного свойства, нельзя показать, что свойство может присутствовать не у всех экземпляров объектов, нет понятия агрегированного объекта, нет изобразительного средства для отображения альтернативной связи («арк»). Все это нужно отобразить, пользуясь имеющимися в наличии изобразительными средствами.

При построении ER-модели в Design/IDEF предлагается придерживаться следующих рекомендаций*.

Если простой объект имеет несколько возможных идентификаторов, то из них следует выбрать тот, который будет использоваться в качестве первичного ключа таблицы, и описать его как **Primary Key.** Рекомендации по выбору первичного ключа реляционной таблицы изложены в разд. 3.3.

Для остальных возможных идентификаторов надо определить, есть ли необходимость проверять их уникальность в процессе ведения базы данных. Те идентификаторы, для которых это необходимо делать, нужно обозначить как **Alternate Key (AK).**

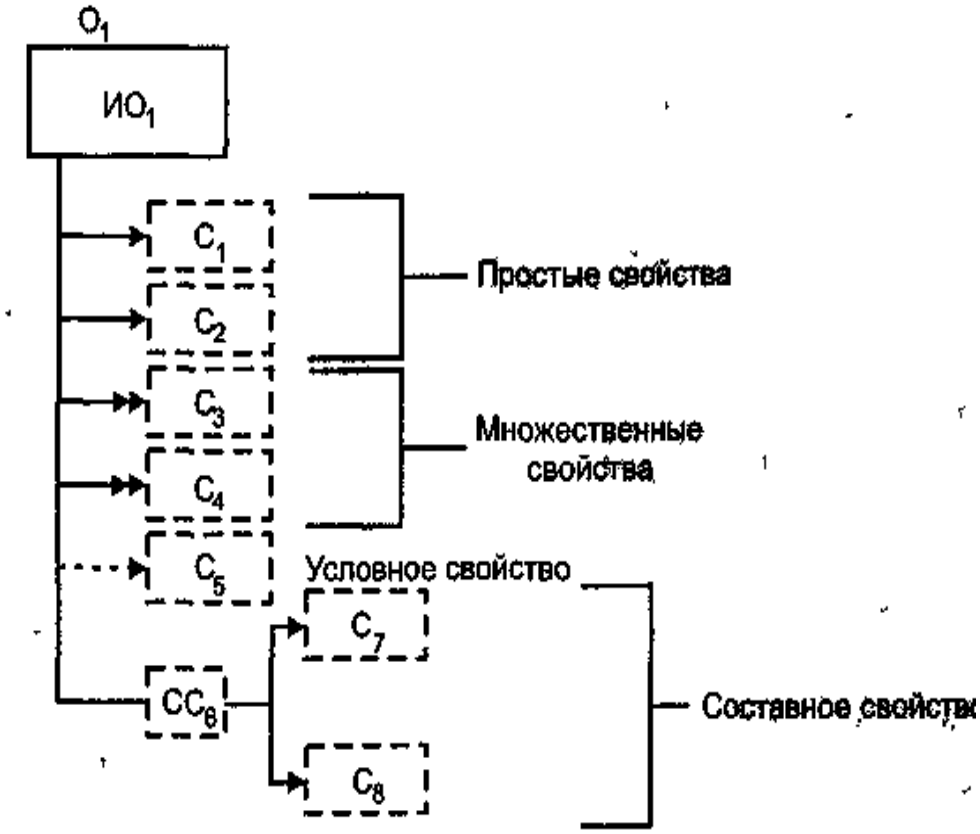
Если объект имеет неуникальное имя (например, ФИО для сущности ЛИЧНОСТЬ), то его следует описать как **Inversion Entry (IE),** поскольку имя часто используется для поиска. В качестве инверсных входов могут быть описаны и другие атрибуты. Для того чтобы определить, для каких атрибутов нужно задавать свойство **Inversion Entry,** необходимо проанализировать характер запросов к создаваемой таблице: если по данному атрибуту ожидается частый выборочный поиск или требуется сортировка, то этот атрибут следует описать как инверсный вход. Так как альтернативных ключей и инверсных

---

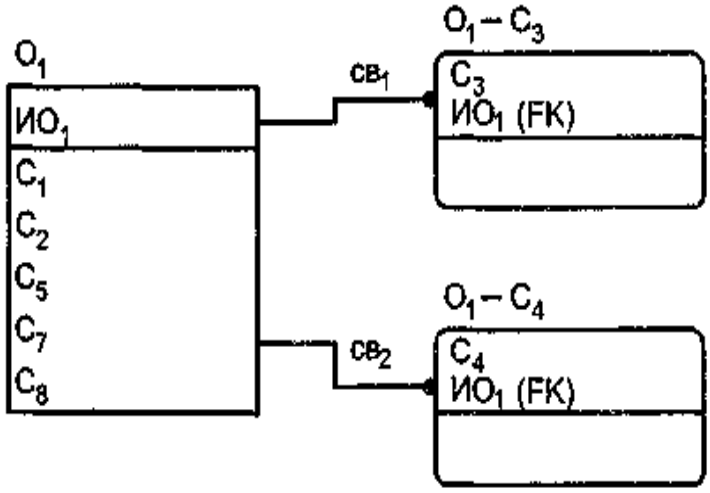
* Поскольку сущности в методологии IDEF фактически соответствуют таблицам реляционной базы данных, то целесообразно сначала изучить разд. 3.3, тогда рекомендации, излагаемые в данном разделе, будут более понятными.

входов может быть несколько, то при описании соответствующего атрибута надо указать порядковый номер **АК** или **IE**.

При наличии у объекта множественных свойств нужно каждому из множественных свойств поставить в соответствие отдельную сущность. Название множественного свойства следует определить как ключевой атрибут. Затем надо связать идентифицирующей связью, направленной от основной сущности ко вновь созданной сущности, эти элементы модели (рис. 2.62).



а



б

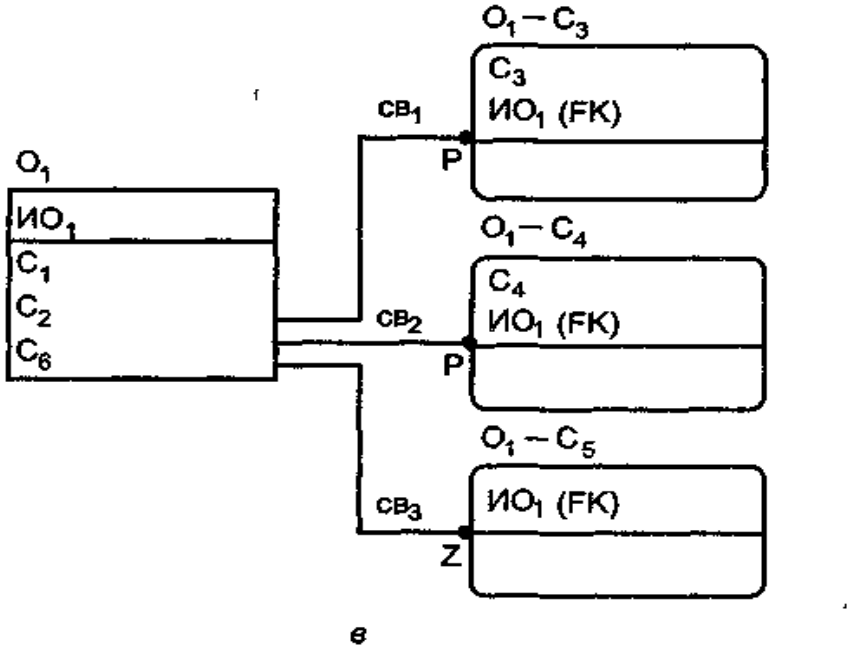


Рис. 2.62. Отображение единичных, множественных и составных свойств (переход от базовой модели к IDEF1X:
а - простой объект (базовая ER-модель);
б - вариант 1 отображения объекта в IDEF1X; *в* - вариант 2

Поскольку в Design/IDEF нет возможности отображать составные свойства, то проектировщик должен принять решение, как это составное свойство будет храниться в базе данных. В зависимости от принятого решения нужно либо описать это составное свойство как один атрибут, либо каждый из составляющих его элементов описать как отдельный атрибут. На рис. 2.62, б изображено второе из названных решений. Если использовать первое решение, то вместо атрибутов C_7 , C_8 следовало бы описать атрибут C_6 (рис. 2.62, в).

Отсутствие понятия «условное свойство» приводит к сложностям при моделировании предметной области. Возможны несколько вариантов выхода из сложившейся ситуации.

1. Никак в ER-модели не отражать, что свойство условное, и описывать его как обычный атрибут (такое решение изображено на рис. 2.62,6).

2. В соответствие условному свойству ставится отдельная сущность, которая идентифицирующей связью соединяется с основной сущностью. При этом тот факт, что свойство является условным, передается путем выбора соответствующего типа кардинальности связи (такое решение изображено на рис. 2.62, *в).*

3. Объект, содержащий условные свойства, отображать как обобщенный. Каждому условному свойству будет соответствовать категория объектов. Этот вариант использован на рис. 2.61, когда условному свойству «Звание_ученое» был поставлен в соответствие видовой объект ИМЕЮЩИЕ_ЗВАНИЕ.

Каждый из вариантов имеет свои недостатки. Выбор варианта будет зависеть от выбранного проектного решения по структуре БД.

Для отображения обобщенного объекта в Design/IDEF имеются специальные изобразительные средства. При изображении обобщенного объекта то свойство, по которому проводится разбиение класса на подклассы, обозначается как дискриминатор. Каждому дискриминатору на схеме соответствует специальный знак. После этого каждому подклассу ставится в соответствие отдельная сущность, в которой перечисляются атрибуты, присущие этому подклассу. Ключ видовых сущностей при описании указывать не нужно. Далее от дискриминатора к видовым сущностям протягивается связь. При этом происходит автоматическая миграция ключа. Подчиненные сущности становятся, таким образом, зависимыми от идентификации сущностями (рис. 2.63).

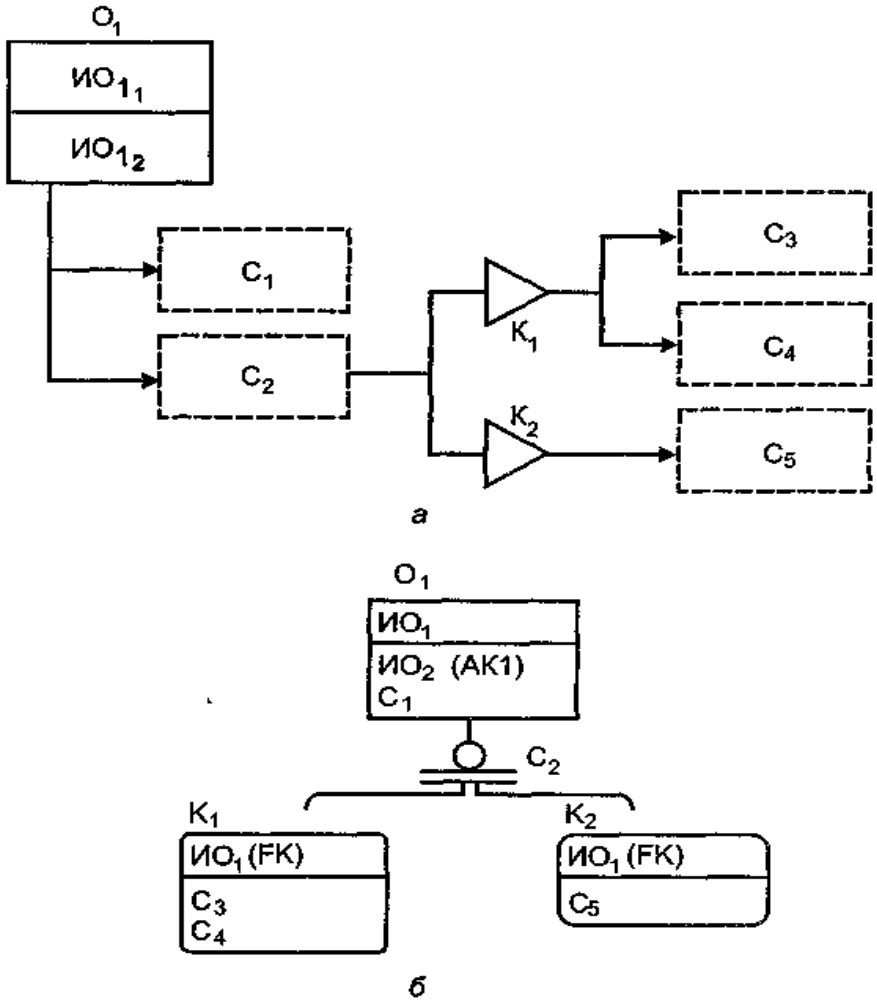


Рис. 2.63. Изображение обобщенного объекта: *а* - фрагмент базовой ER-модели; *б* - отображение в IDEF1X

В Design/IDEF можно отобразить многоаспектную и многоуровневую классификацию объектов.

Для изображения агрегированных объектов в Design/IDEF не предусмотрено никаких специализированных изобразительных средств. Агреги-

рованные объекты (процессы) в Design/IDEF следует отображать следующим образом (рис.2.64):

1)создать сущность, соответствующую данному объекту;

2)соединить ее идентифицирующими связями с сущностями, участвующими в данном процессе;

3)если для полной идентификации изображаемой сущности мигрировавших ключей оказывается недостаточно, то дополнительные атрибуты при их описании нужно задать как первичный ключ;

4) описать остальные атрибуты.

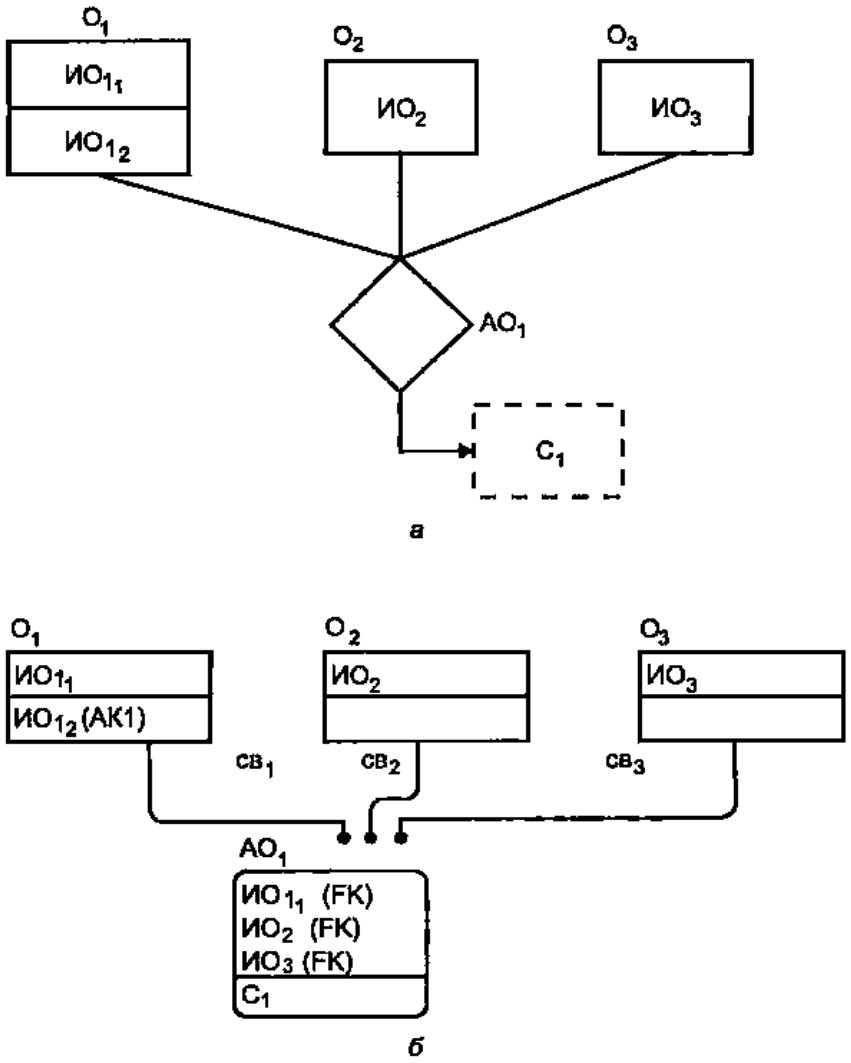


Рис. 2.64. Отображение агрегированного объекта
в базовой ER-модели *(а)* и в IDEF1X *(б)*

На рис. 2.65 изображен агрегированный объект УСПЕВАЕМОСТЬ. Атрибут «Дата» описан как первичный ключ. «Код_студента» и «Код_предмета» мигрируют в сущность УСПЕВАЕМОСТЬ при установлении соответствующих связей.

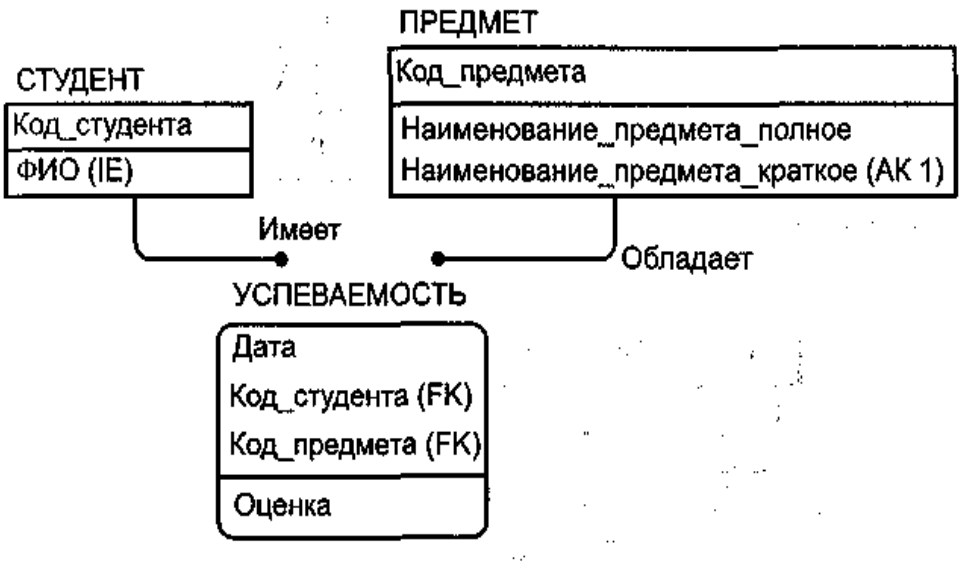


Рис. 2.65. Агрегированный объект УСПЕВАЕМОСТЬ

Из вышеизложенного следует, что механизм установления идентифицирующей связи используется часто, и не только в случаях, когда в предметной области реально существует такая идентификация (например, когда нумерация участков ведется внутри каждого цеха и в тому подобных ситуациях), но и при обходе ограничений Design/IDEF, когда приходится вводить дополнительные объекты, или при изображении агрегированных объектов в виде обычной сущности в модели Design/IDEF.

Для определения связи в Design/IDEF надо комбинировать возможности, задаваемые в **Relationship Type** и **Relationship Cardinality.**

Идентифицирующая (Identifying) и неидентифицирующая (Non-Identifying) связи в общем случае соответствуют отношению 1:M. Если надо указать, что связь 1:1, то можно для Relationship Cardinality этой связи указать Exactly 1 (рис. 2.66).

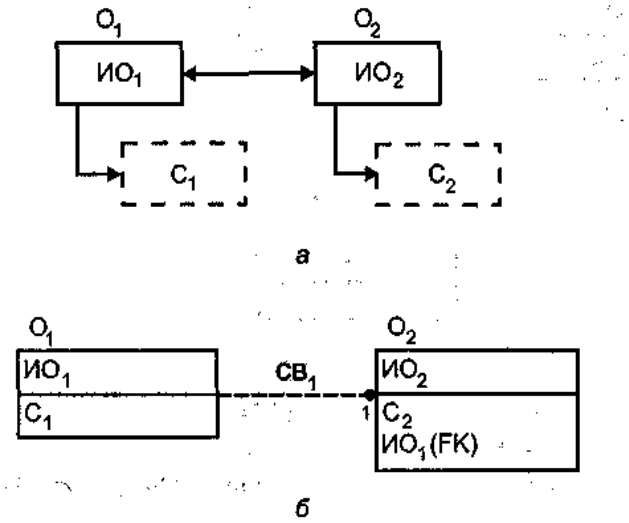


Рис. 2.66. Отображение связи 1:1 в базовой модели *(а)* и в IDEF1X*(б)*

Множественная связь называется в Design/IDEF *неспецифической* (Non-Specific) (рис. 2.67). Так как алгоритм проектирования БД в Design/IDEF не обеспечивает автоматического преобразования связей M:M, то перед генерацией описания БД необходимо устранить неспецифические связи и преобразовать их в специфические. Для этого нужно ввести дополнительную связующую сущность. Никаких атрибутов для нее определять не надо. Далее следует связать вновь введенную сущность с ранее существовавшими объектами идентифицирующей связью. В результате получится схема, изображенная ни рис. 2.67, *в*.

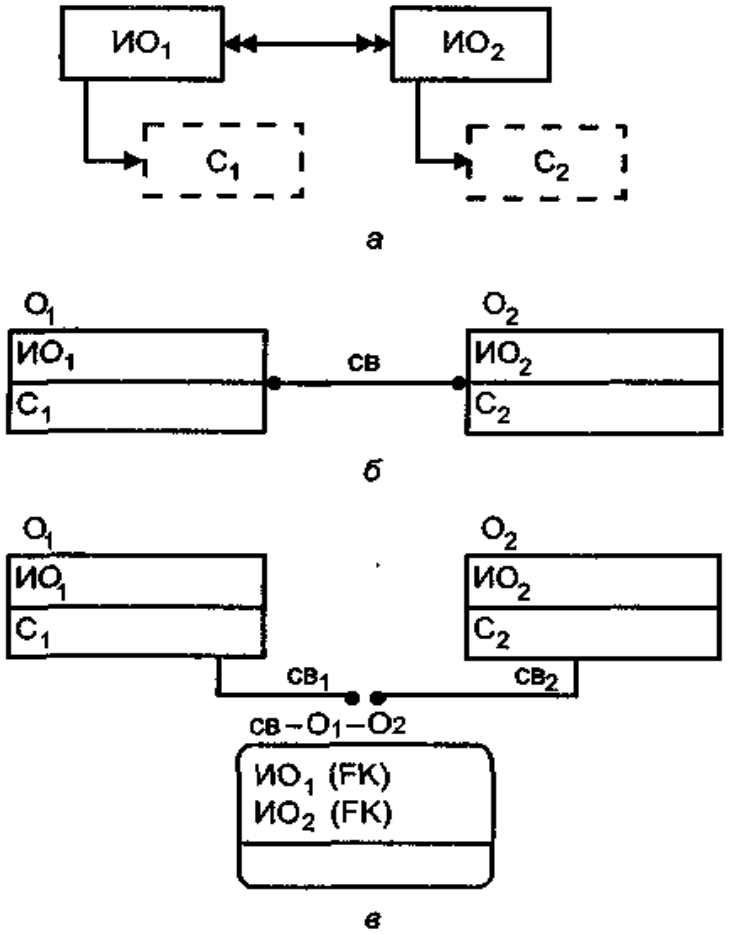


Рис. 2.67. Отображение связи M:M в базовой модели (а);
в IDEF1X - неспецифическая связь (б);
преобразование неспецифической связи в специфические (*в*)

Соответствие базовой модели и модели Design/IDEF при изображении идентифицирующей связи отображено на рис. 2.68, а неидентифицирующей - на рис. 2.69.

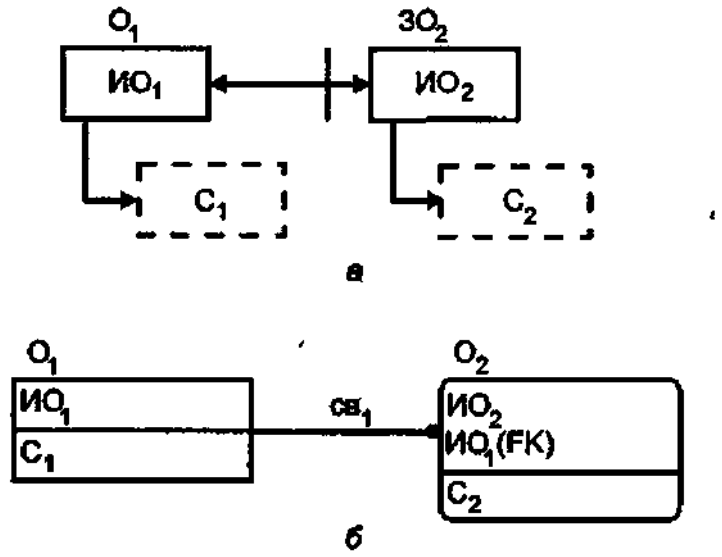


Рис. 2.68. Отображение идентифицирующей связи в базовой модели (а) и в IDEF1X (б)

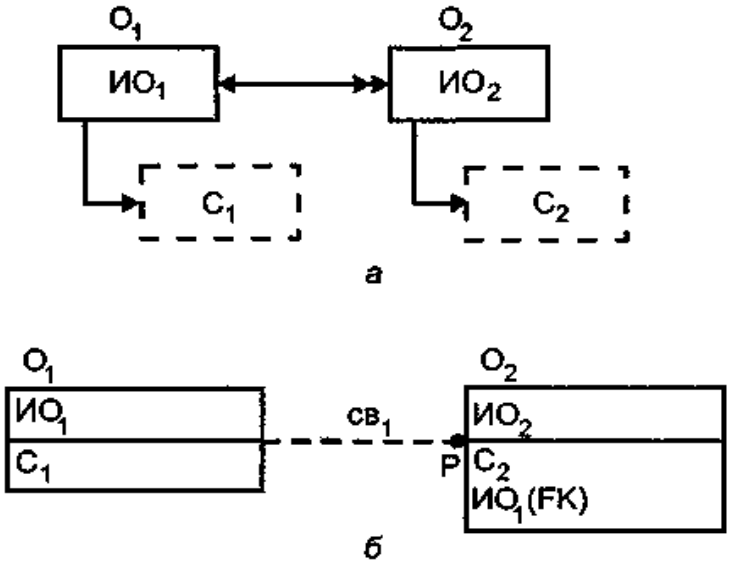


Рис. 2.69. Отображение связи 1:М (неидентифицирующая связь, обязательный класс членства с обеих сторон) в базовой модели *(а)* и в IDEFIX(*б*)

В базовой модели для характеристики связи используются три независимые характеристики:

1) тип связи (один к одному (1:1), один ко многим (1:М) и многие ко многим (М:М));

2) класс принадлежности (обязательный и необязательный);

3) зависимость по идентификации (зависимая и не зависимая по идентификации сущности).

В Design/IDEF выделяют две группы характеристик: **Relationship Type** и **Relationship Cardinality.** В связи с этим не все сочетания, которые можно передать в базовой модели, удается отобразить в Design/IDEF. На рис. 2.70

представлено соответствие конструкций базовой модели и Design/IDEF1X при изображении типа связей и класса принадлежности. В Design/IDEF при задании неспецифической связи кардинальность связи указать нельзя. Поэтому в Design/IDEF нельзя отобразить ситуации, когда при связи M:M наблюдается необязательный класс принадлежности со стороны одной из сущностей или со стороны обеих сущностей. Кроме того, в Design/IDEF нет возможности отобразить класс принадлежности сущностей, к которым направлена связь.

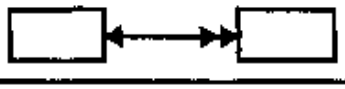
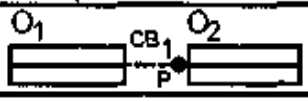
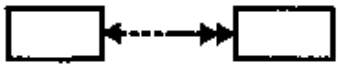
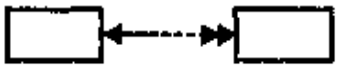
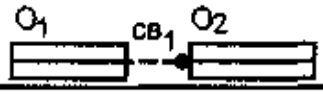
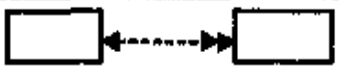
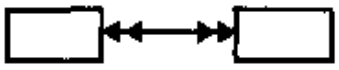
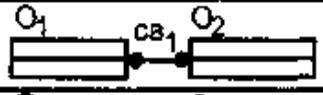
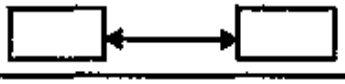
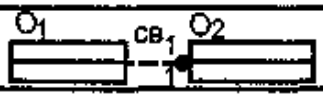
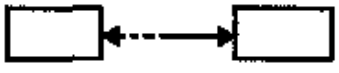
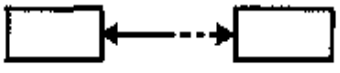
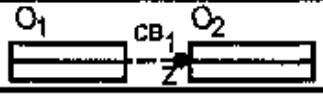
N п/п	Изображение в базовой модели	Изображение в Design/IDEF		
		Relationship Type	Relationship Cardinality	Графическое изображение
1		Non-Identifying	One or Many [P]);	
2		Класс членства сущности, находящейся в конце связи, указать нельзя		
3		Non-Identifying	(Zero, One or Many);	
4		Класс членства сущности, находящейся в конце связи, указать нельзя		
5		Non-Specific	Нельзя указать	
6		Non-Identifying	Exactly 1	
7		Класс членства сущности, находящейся в конце связи, указать нельзя		
8		Non-Identifying	Zero or One [Z]	

Рис. 2.70**.** Типы связи и класс членства. Соответствие базовой модели и Design/IDEF1X

На построение ER-модели оказывает влияние алгоритм проектирования базы данных. В Design/IDEF каждой сущности ER-модели соответствует таблица в реляционной базе данных. Поэтому необходимо уже на стадии ER-моделирования решить, каким сущностям ставить в соответствие таблицы, а каким - нет, и в качестве объектов ER-модели изображать только те сущности, которым будут соответствовать отдельные таблицы. Например, «дату» не нужно отображать как отдельную таблицу, поэтому эту сущность следует из модели убрать. Все атрибуты или сущности, соответствующие вычисляемым значениям, которые проектировщик не хочет хранить в базе данных, тоже должны быть устранены из ER-модели.

В Design/IDEF можно нарисовать несколько связей между парой объектов. При этом на схеме миграция ключа происходит только один раз, а не столько раз, сколько связей объявлено между парой объектов (рис. 2.71).

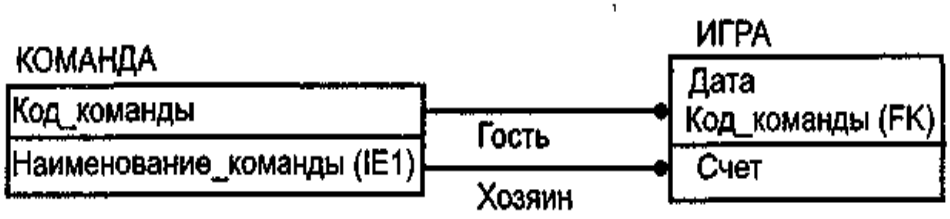


Рис. 2.71. Изображение нескольких связей между парой сущностей

На самом деле, если посмотреть на соответствующее сгенерированное системой описание объекта на SQL, в нем присутствуют оба атрибута связи:

CREATE TABLE игра
(дата **DATE NOT NULL,**
счет **CHAR NULL,**
код_команды **CHAR NOT NULL,**
код_команды **CHAR NOT NULL).**

Таким образом, в системе имеется некоторое несоответствие между графическим и аналитическим представлением сущности. Следует также обратить внимание на некоторые неточности в полученном описании. Так, двум полям в таблице «Игра» дано одинаковое имя «код_команды», что недопустимо.

В связи с тем, что ER-модели играют несколько разных ролей в процессе проектирования информационных систем, среди которых важнейшей является «адекватное отображение предметной области и использование ER-модели для общения между всеми участниками (как проектировщиками, так и заказчиками) процесса создания ИС», при применении CASE-средств, не обладающих развитым многовариантным алгоритмом проектирования, рекомендуется создавать по меньшей мере две модели:

- исходную, описывающую предметную область безотносительно к заложенному в CASE-системе алгоритму преобразования ER-модели в логическую модель целевой базы данных;
- ER-модель, которая будет использоваться для генерации схемы базы данных.

Поскольку Design/IDEF не производит преобразование имен сущностей и атрибутов в соответствии с требованиями целевой СУБД, то в модели, предназначенной для генерации схемы базы данных, имена сущностям и атрибутам следует давать такие, которые допустимы в целевой СУБД.

В связи с тем, что изобразительные средства ER-моделирования в Design/IDEF бедны, процесс моделирования предметной области не является естественным. Практически специалист, строящий ER-модель в среде Design/IDEF, должен выполнить проектирование структуры реляционной базы данных вручную. Подобные средства способствуют решению некоторых проблем, стоящих при создании и развитии ИС, но не облегчают задачу проектирования структуры базы данных.

2.6. Особенности моделирования в ERWin

2.6.1. Общие замечания

Уточнение терминологии

В ERWin, как, впрочем, и в некоторых других CASE-системах, применяется терминология, отличающаяся от традиционно используемой в теории баз данных. Так, понятие «физическая модель» традиционно использовалось в литературе по БД для описания способа хранения данных в запоминающей среде. В ERWin *физической моделью* называется описание (логической) структуры базы данных в среде выбранной целевой СУБД.

Нотации, используемые при построении концептуальной модели

Для создания новой модели после запуска системы нужно выбрать позицию меню File/New.

При изображении ER-модели в ERWin можно выбрать нотацию, которая будет использоваться. Для того чтобы осуществить переключение между нотациями, следует выбрать позиции меню **Options/ Preferences** и в появившемся окне **Preferences** выбрать вкладку **Methodology.** Как видно из рис. 2.72, для логического представления ER-модели можно осуществлять выбор из двух нотаций: IDEF1X (Integration DEFinition for Information Modeling) и IE (Information Engineering).

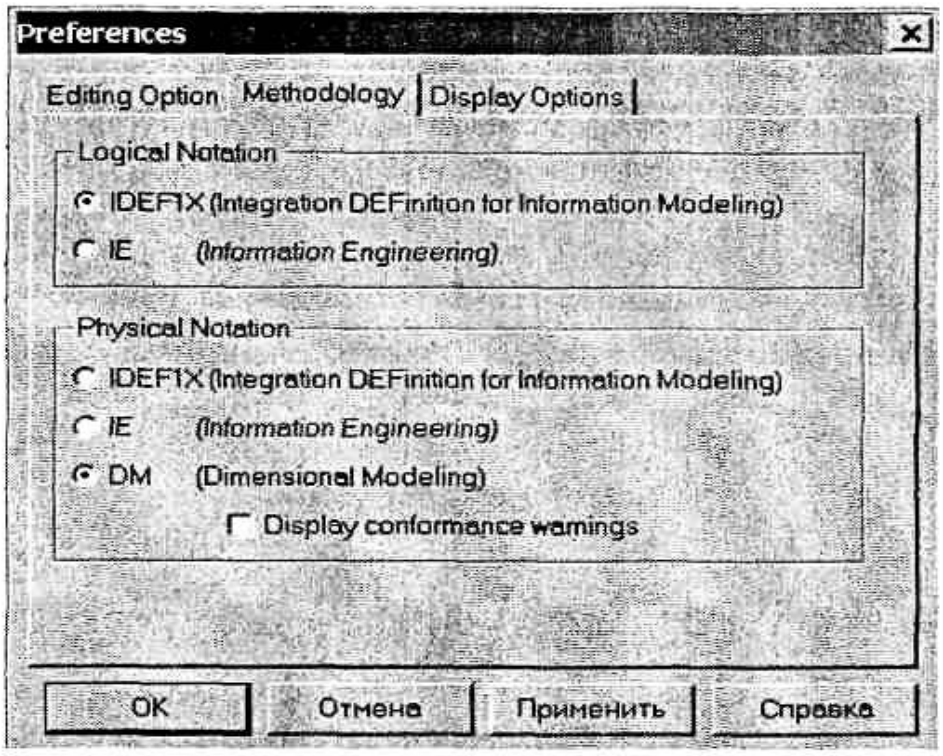


Рис. 2.72. Выбор нотации для представления логической модели

Нотация IDEF1X является стандартной и, по сути, не отличается от соответствующей нотации, рассмотренной при описании Design/ IDEF.

Чтобы панель инструментов (ERWin Toolbox) для случая использования нотации IDEF1X появилась на экране, надо выбрать позицию меню **Window/Toolbox** (рис. 2.73).

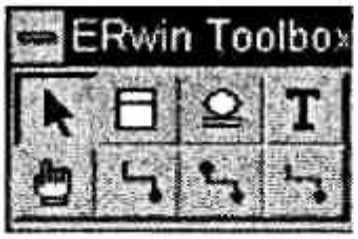


Рис. 2.73. Вид панели инструментов
для случая использования нотации IDEF1X

Реализации систем, естественно, несколько отличаются друг от друга. Так, если в Design/IDEF вид связи задается при ее описании, то в ERWin имеются три кнопки  , которые соответствуют идентифицирующей связи, связи «многие ко многим» и неидентифицирующей связи (в порядке их расположения на панели). Кнопка  используется при создании обобщенного объекта.

Вид панели элементов для случая использования нотации IE представлен на рис. 2.74. Различия в этих нотациях относятся к классу несущественных (*см.* разд. 2.3). Разница заключается в том, что для обозначения множественного конца линии связи в нотации IDEF1X используется точка, а в нотации IE - «лапка», а также используется другой графический символ  при отображении обобщенной сущности.

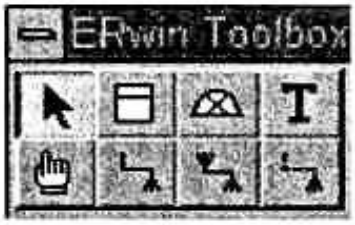


Рис. 2.74. Вид панели инструментов
для случая использования нотации IE

Модель, построенная в одной нотации, может быть автоматически преобразована в другую нотацию. Для этого следует опять вернуться в окно **Preferences** (*см.* рис. 2.72), выбрав меню **Options/ Preferences,** и выбрать на вкладке **Methodology** другую методологию.

Далее рассмотрим создание логической модели в нотации IDEF1X.

2.6.2. Построение логической модели

Создание новой сущности

Для создания новой сущности следует воспользоваться кнопкой  . После позиционирования создаваемой сущности на экране следует перейти в Редактор сущности (**Entity Editor**) и задать требуемые характеристики (рис. 2.75). На последующую генерацию схемы базы данных окажут влияние имя сущности (Name) и признак Logical Only. Сущность, отмеченная как Logical Only, не будет переноситься в физическую модель.

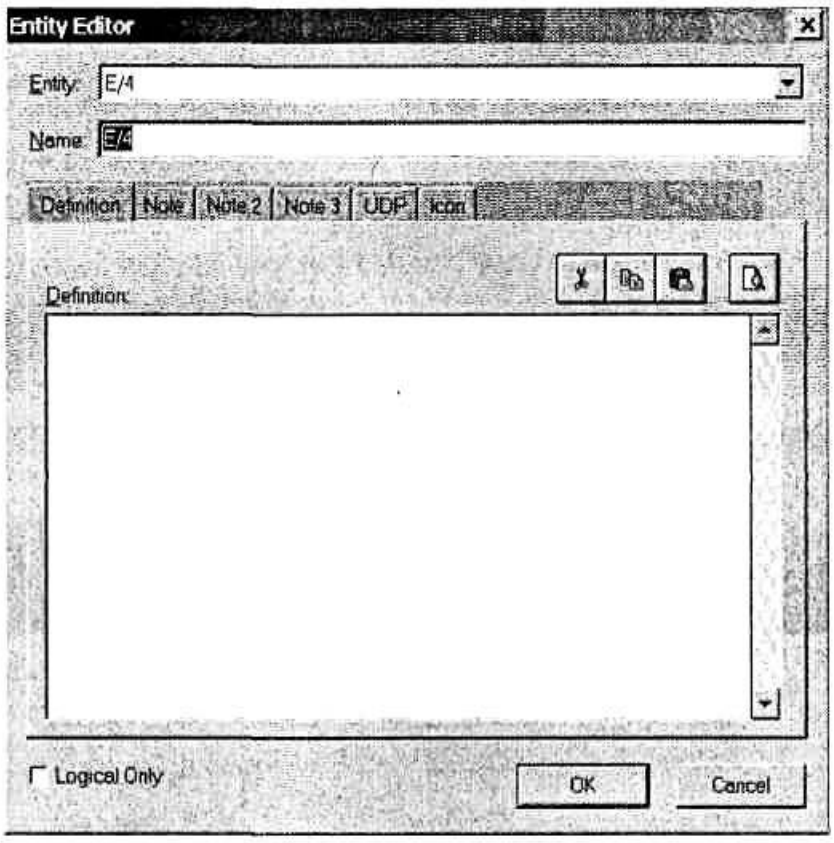


Рис. 2.75. Вид окна описания сущности

Вся остальная информация, фиксируемая при описании объекта, безусловно, важна для общего понимания модели, ее документирования, но прямого отношения к проектированию баз данных не имеет и потому подробно рассматриваться не будет.

Описание свойств сущности

После описания сущности следует перейти к описанию ее свойств (атрибутов). Для этого следует перейти в Редактор атрибутов **(Attribute Editor)** - появится соответствующее окно (рис. 2.76).

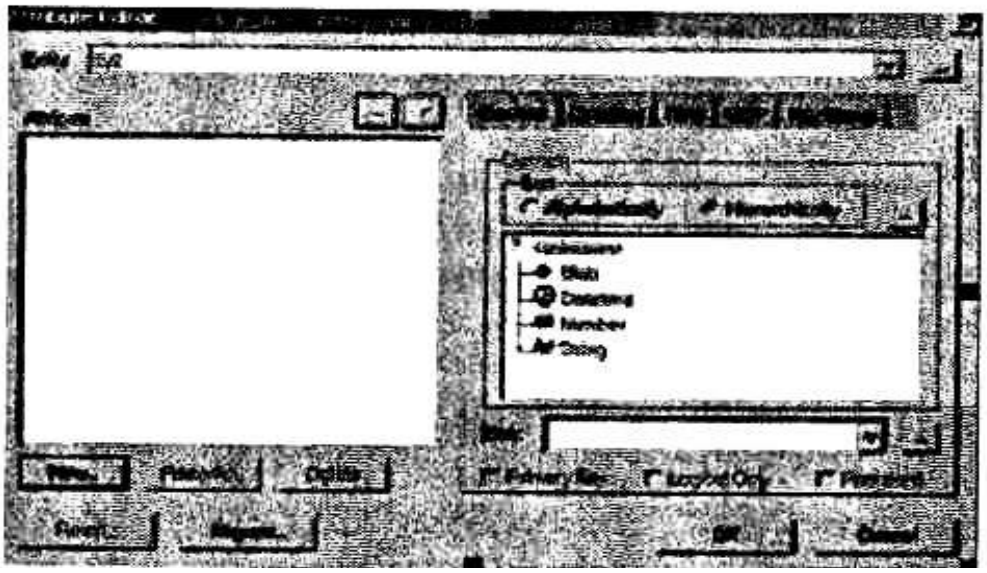


Рис. 2.76. Вид окна Редактор атрибутов

Для создания нового атрибута надо нажать кнопку **New,** после чего появится окно (рис. 2.77) **New Attribute** (Новый атрибут).

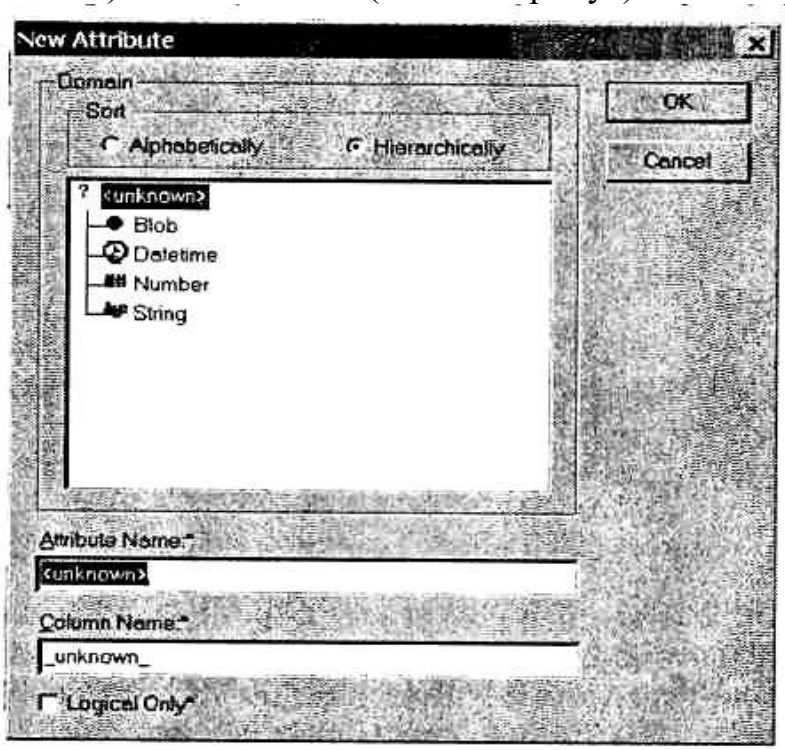


Рис. 2.77. Начальный вид окна Новый атрибут

При описании нового атрибута можно задать для него два имени: имя, которое будет указываться в логической модели (Attribute Name), и имя, используемое в физической модели (Column Name). Если вручную не задавать Column Name, то при переходе к физической модели в зависимости от выбранной целевой СУБД Attribute Name будет автоматически преобразован в Column Name в соответствии с ограничениями СУБД на имена (длина, допустимость пробелов).

Кроме того, следует соотнести атрибут с доменом. Эта информация будет использоваться для определения типа данных при переходе к физической модели.

При описании атрибута можно поставить признак **Logical Only** (только логический). Это никак не скажется на логической модели, но окажет влияние на переход от логической к физической модели: атрибуты, отмеченные как **Logical Only,** не будут переноситься в физическую модель.

Для атрибутов, соответствующих первичным ключам таблицы, следует установить признак **Primary Key** (рис. 2.78).

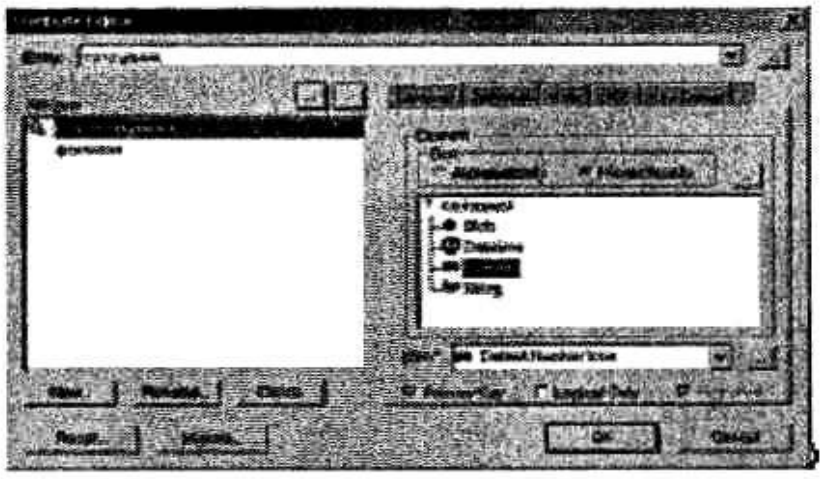


Рис. 2.78. Задание первичного ключа

Кроме того, для атрибута можно задавать свойство **Required** (обязательный). Для атрибута, выбранного первичным ключом, это свойство является неактивным (*см.* рис. 2.78), поскольку свойство обязательности и так (по определению ключа) присуще элементам ключа.

Дополнительные свойства атрибутов

При создании реляционных баз данных по отдельным полям (совокупностям полей) могут быть построены индексы. По умолчанию по полям, выбранным в качестве первичного ключа, строится уникальный индекс. По полям, которые являются альтернативными ключами таблицы, также можно построить уникальные индексы. По тем полям, по которым часто осуществляется выборочный поиск, для ускорения поиска также целесообразно провести индексирование. Если поле, по которому проводится индексирование, является неуникальным, то оно называется *инверсным входом* (Inversion Entry).

Как отмечалось ранее при общей характеристике инфологического моделирования, понятия ключа (первичного, альтернативного, внешнего) и индексирования не присущи предметной области. Понятие ключа относится к категориям реляционной модели данных, а индексирование — это способ логического упорядочения данных. Однако в ERWin и некоторых других

CASE-системах эти вопросы решаются на стадии концептуального проектирования. Для того чтобы задать необходимость индексирования, можно, позиционируя на сущности, нажать правую кнопку мыши и в появившемся контекстном меню (рис. 2.79) выбрать позицию **Key Group Editor.**

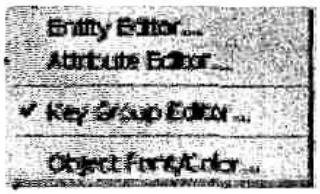


Рис. 2.79. Контекстное меню сущности

Предположим, что мы хотим задать инверсный вход для поля Фамилия в таблице «Сотрудник». В этом случае в окне **Key Group Editor** (рис. 2.80) нужно щелкнуть по кнопке **New**. В появившемся окне **New Key Group** следует выбрать переключатель **Inversion Entry** и щелкнуть по кнопке **ОК**.

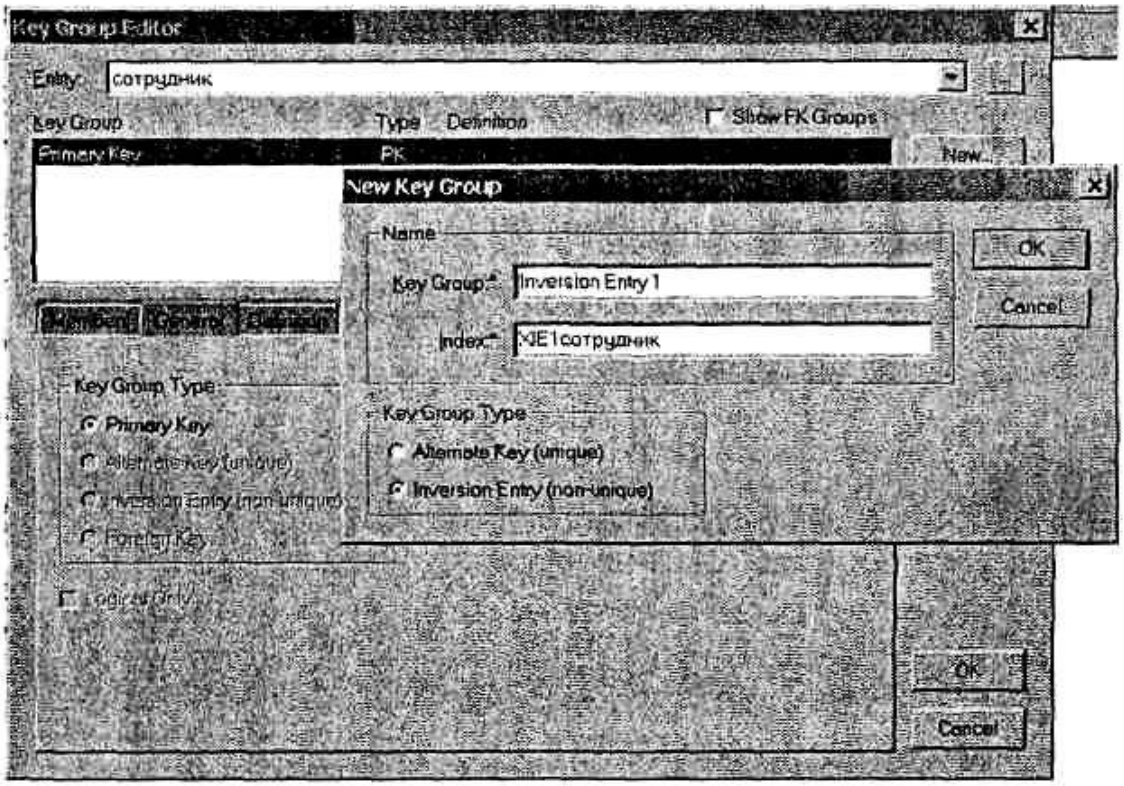


Рис. 2.80. Задание инверсного входа. Экран 1

После этого в окне **Available Attributes** нужно выбрать поле, которое определяется как инверсный вход, и перенести его в окно **Key Group Members** (рис. 2.81).

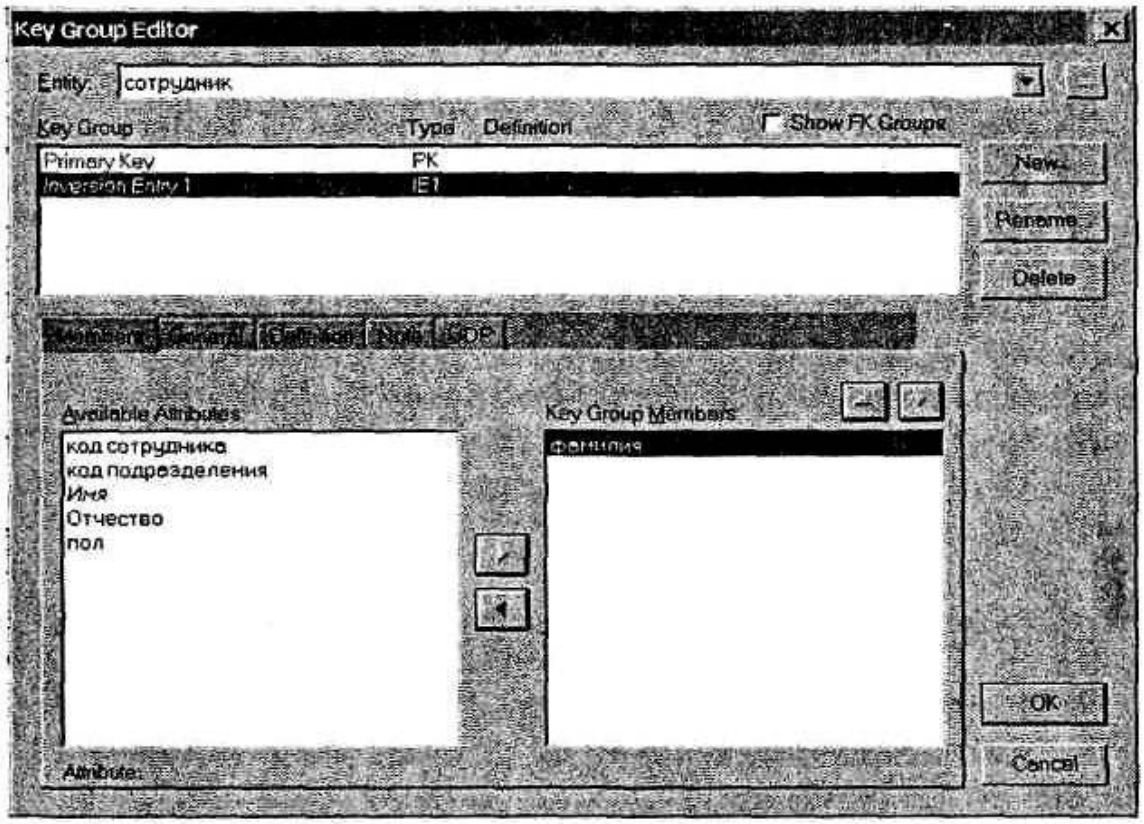


Рис. 2.81. Задание инверсного входа. Экран 2

Первичные и альтернативные ключи, инверсные входы можно задать и иным способом. Для этого в окне Редактор атрибутов (**Attribute Editor**) нужно выбрать вкладку **Key Group** (рис. 2.82), щелкнуть по кнопке ..., после чего опять появится окно **Key Group Editor**.

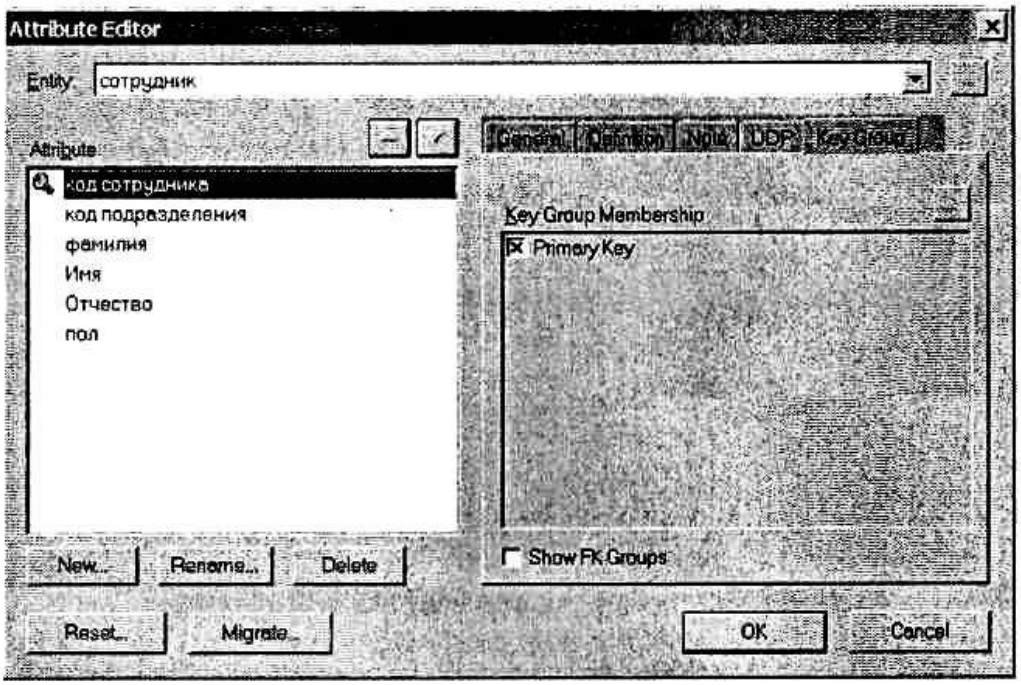


Рис. 2.82. Окно описания атрибутов. Вкладка **Key Group**

Если требуется провести индексацию по совокупности полей, т.е. создать составной индекс, то в **Key Group Members** нужно включить несколько полей (рис. 2.83).

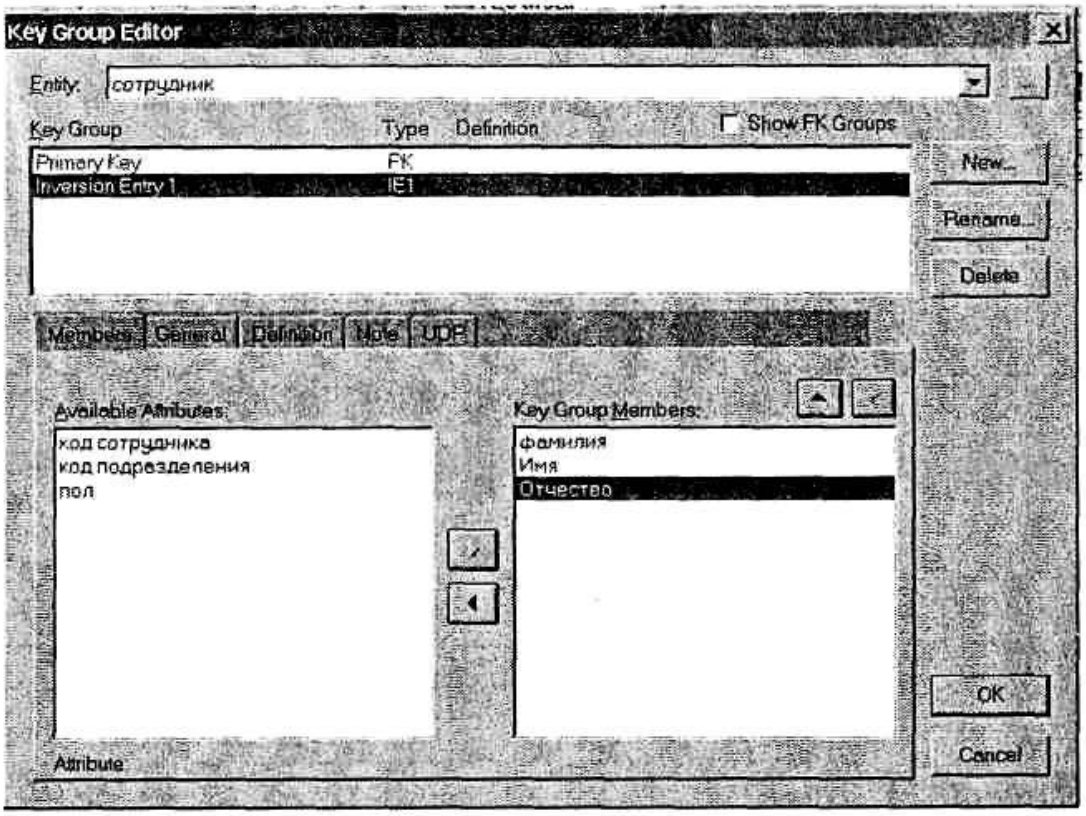


Рис. 2.83. Создание составного индекса

В ERWin ключи (первичные и альтернативные) и индексирование можно задавать при описании логической модели, а можно - и при описании физической модели (*см.* разд. 3.4). При переключении между физической и логической моделью вновь введенные характеристики будут отображаться в обоих представлениях безотносительно к тому, в каком режиме они были введены.

Описание обобщенных объектов

В ERWin можно изображать обобщенные объекты. Для этого следует предварительно изобразить как родовой, так и видовые объекты. Далее нужно на панели элементов выбрать кнопку  , после чего щелкнуть левой кнопкой мыши по родительской, а затем по видовой сущности. Для установления связи со следующим видовым объектом надо сначала щелкнуть левой кнопкой мыши на схеме по символу категории, а затем - по следующей видовой сущности.

На рис. 2.84 изображен фрагмент модели, на котором для сущности СОТРУДНИК выделены две категории: ШТАТНЫЕ и СОВМЕСТИТЕЛИ.

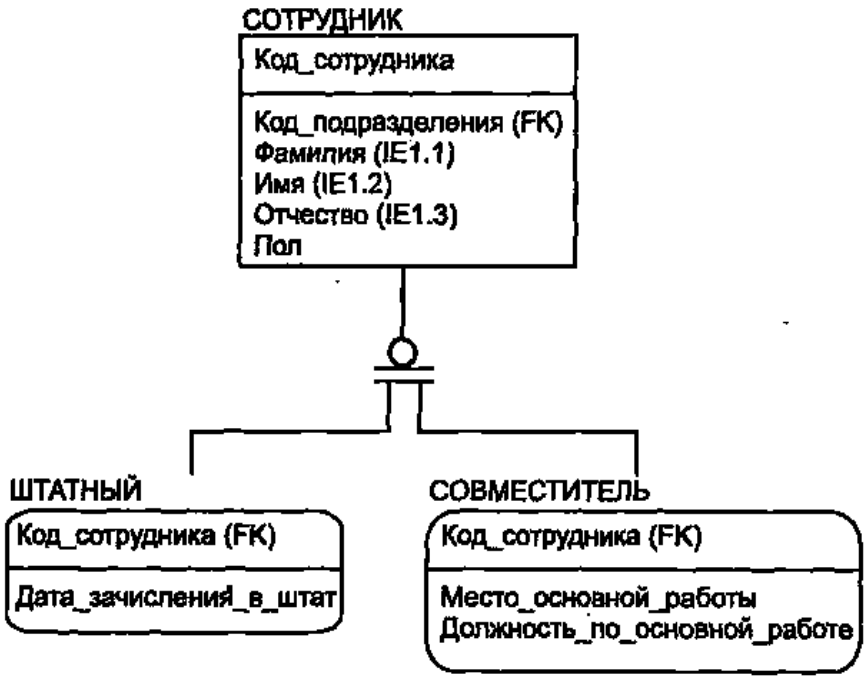


Рис. 2.84. Изображение обобщенного объекта

Обобщенный объект называют *иерархией категорий.* Различают полные (Complete sub-category) и неполные (Incomplete sub-category) категории. На панели элементов имеется только знак полной категории. Чтобы изменить знак категории, необходимо позиционироваться на нем, щелкнуть левой кнопкой мыши и в контекстном меню выбрать позицию **Subtype Relationship.** В появившемся окне (рис. 2.85) следует выбрать переключатель Incomplete.

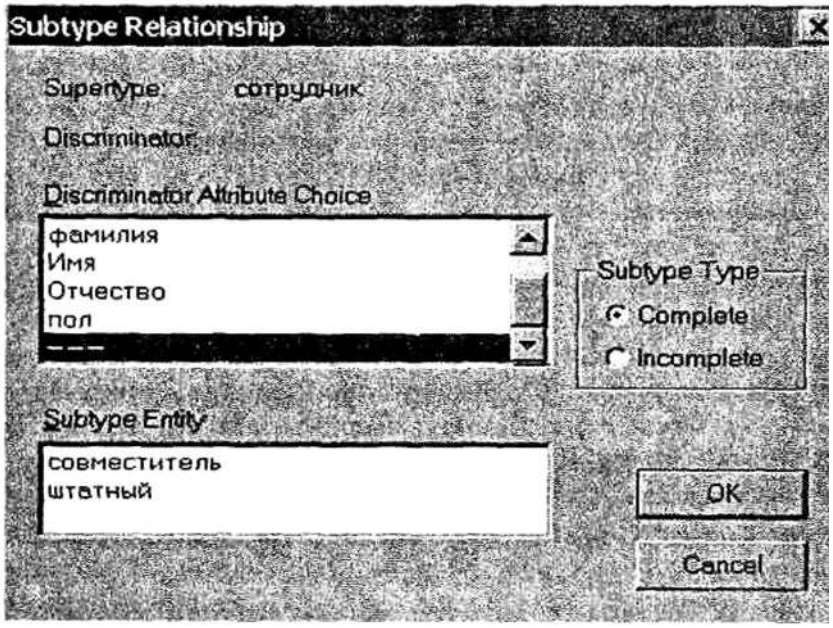


Рис. 2.85. Редактор связи подтипов

Чтобы задать имя признака, по которому проведено деление класса на подклассы, соответствующий атрибут должен быть введен в описание родовой сущности. После этого во внутреннем окне **Discriminator Attribute Choice** окна редактирования связи подтипа следует позиционироваться на этот атрибут (рис. 2.86).

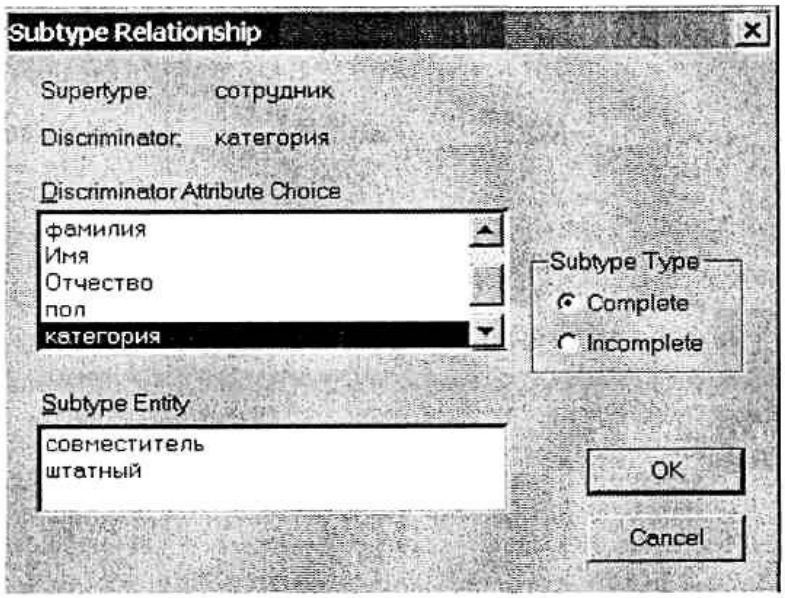


Рис. 2.86. Определение дискриминатора

После выполненных манипуляций фрагмент логической модели, отражающий обобщенный объект СОТРУДНИК, приобрел вид, изображенный на рис. 2.87.

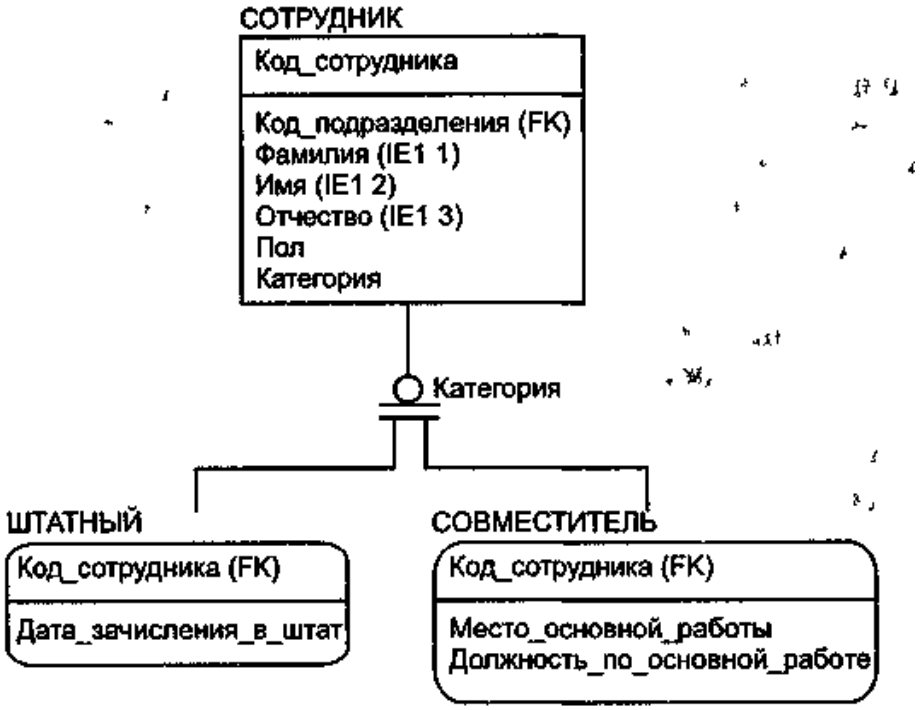


Рис. 2.87. Изображение обобщенного объекта
с заданным именем дискриминатора

Задание связей между сущностями

Виды связей. Прежде чем задавать связь между сущностями, необходимо определить тип этой связи. В ERWin, как и в большинстве других систем, можно задавать только бинарные связи (связи между двумя сущностями). Могут быть заданы:

- идентифицирующая связь «один ко многим» ()
- неидентифицирующая связь «один ко многим» ()
- связь «многие ко многим» ()

Для того чтобы задать связь, нужно выбрать соответствующую этой связи кнопку, после чего щелкнуть кнопкой мыши сначала на родительской, а потом - на дочерней сущности. При задании связи M:M направление связи значения не имеет.

Чтобы отредактировать свойства связи, следует, установив указатель мыши на линии связи, щелкнуть правой кнопкой мыши и в контекстном меню выбрать позицию **Relationship Editor.** В появившемся окне (рис. 2.88) следует выбрать подходящие для данной связи характеристики. Кроме выбора/изменения типа связи (идентифицирующая/неидентифицирующая) можно задать подходящее кардинальное число. Смысл каждого из возможных значений кардинальности понятен из их названий.

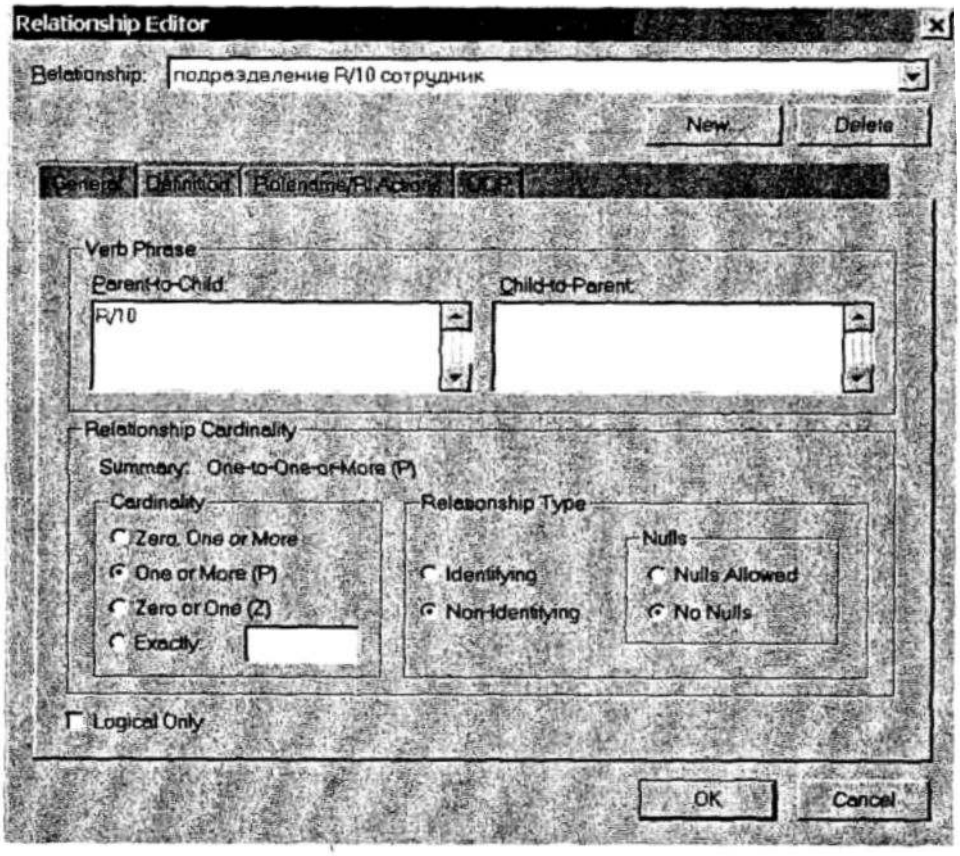


Рис. 2.88. Задание типа и кардинальности связи

Для обозначения типа связи в нотации ERWin/IDEF используются следующие обозначения: идентифицирующая связь изображается сплошной линией, неидентифицирующая - пунктирной. Множественный конец связи обозначается точкой, для единичного конца не используется никакого специального обозначения. Если при описании неидентифицирующей связи выбрана опция **Null Allowed,** то на соответствующем конце связи используется символ ромбика.

Для обозначения кардинального числа используются символы:

- P - один или много;
- Z - нуль или один;
- выбранное при описании связи число - *Exactly.*

По умолчанию символ, определяющий кардинальное число, не отображается на схеме. Для его отображения следует из контекстного меню включить соответствующую опцию (рис. 2.89).

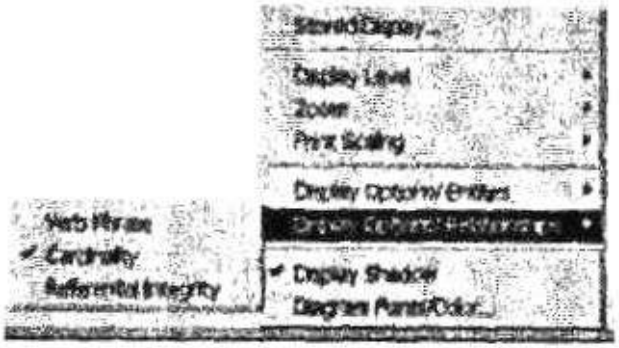


Рис. 2.89. Опции отображения для связей

Примеры изображения связей. На рис. 2.90 отображено описание связи между объектами ПОДРАЗДЕЛЕНИЕ и СОТРУДНИК. В подразделении должен работать хотя бы один сотрудник; идентификатор сотрудника не зависит от идентификации подразделений; сотрудник обязательно приписан к какому-либо подразделению.

Если между объектами задана идентифицирующая связь, то блок **Nulls** является неактивным.

При соединении объектов идентифицирующей связью графическое представление целевой сущности автоматически изменяется (вместо прямоугольника используется фигура с закругленными углами; такая сущность называется *зависимой по идентификации сущностью).*

На рис. 2.91 изображен фрагмент ER-модели, включающий все допустимые типы связи и кардинальности. Связь между ГРУППОЙ и СЛУШАТЕЛЕМ означает, что занятия проводятся либо в группах численностью 10 человек, либо индивидуально. При задании связи между СОТРУДНИКОМ и ЗАГРАН_ПАСПОРТОМ предполагалось, что сотрудник может иметь только один загранпаспорт либо не иметь его вообще.

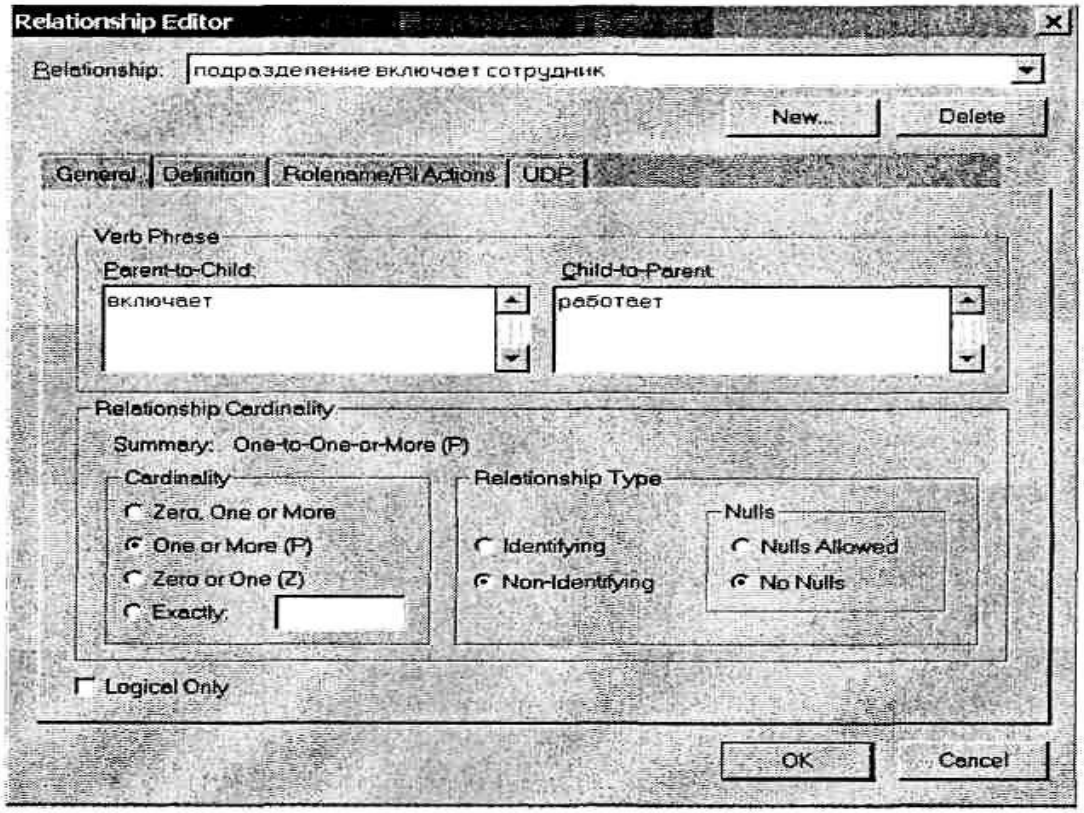


Рис. 2.90. Пример описания неидентифицирующей связи

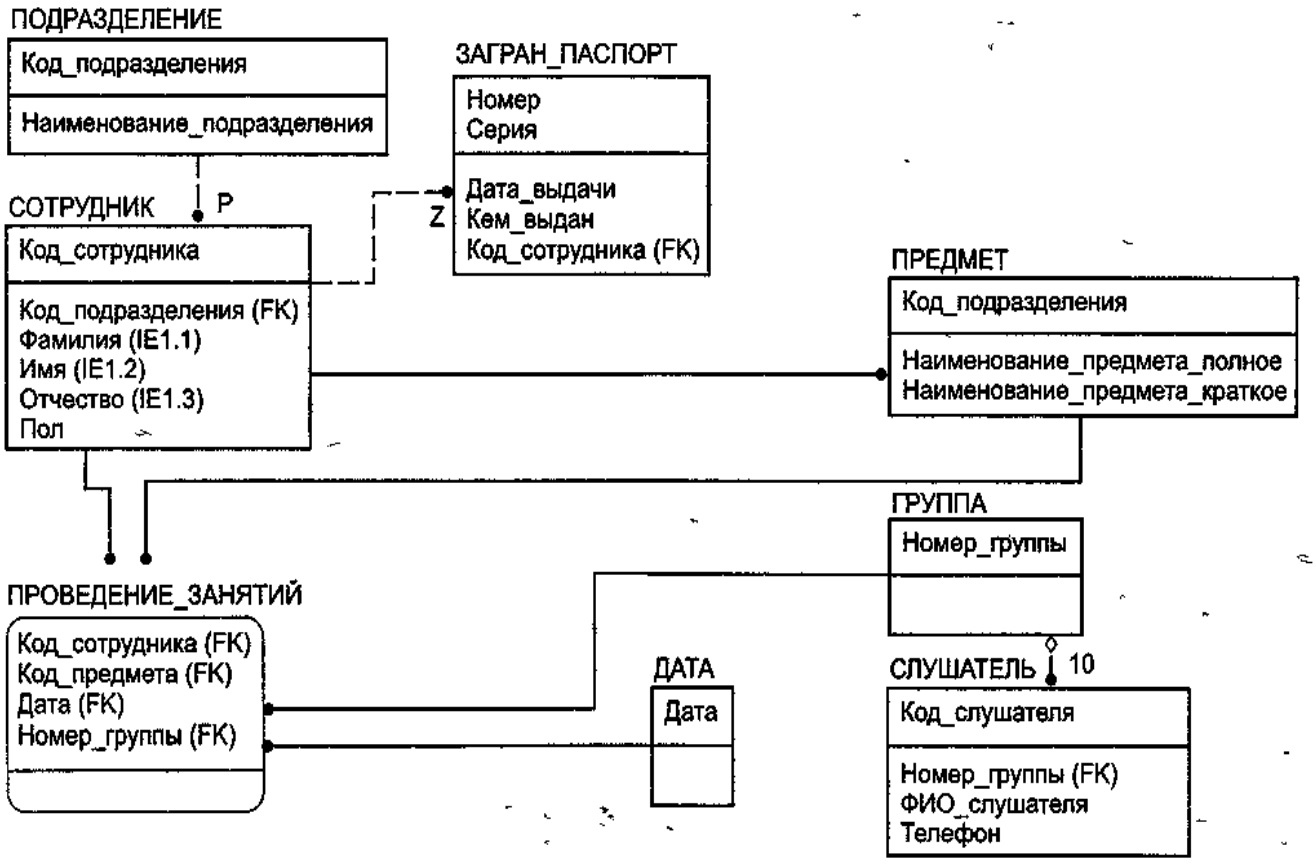


Рис. 2.91. Фрагмент ER-модели. Виды связи

Задание имен связи. В ERWin, так же как и в нашей базовой модели, можно задавать, а можно и не задавать имена связи (**Verb Phrase**). Для того чтобы задать имя связи, необходимо войти в редактор связей (**Relationship Editor**) и на вкладке **General** в блоке **Verb Phrase** в окнах **Parent-to-Child** и **Child-to-Parent** задать имя связи в прямом или обратном направлении соответственно (рис. 2.92).

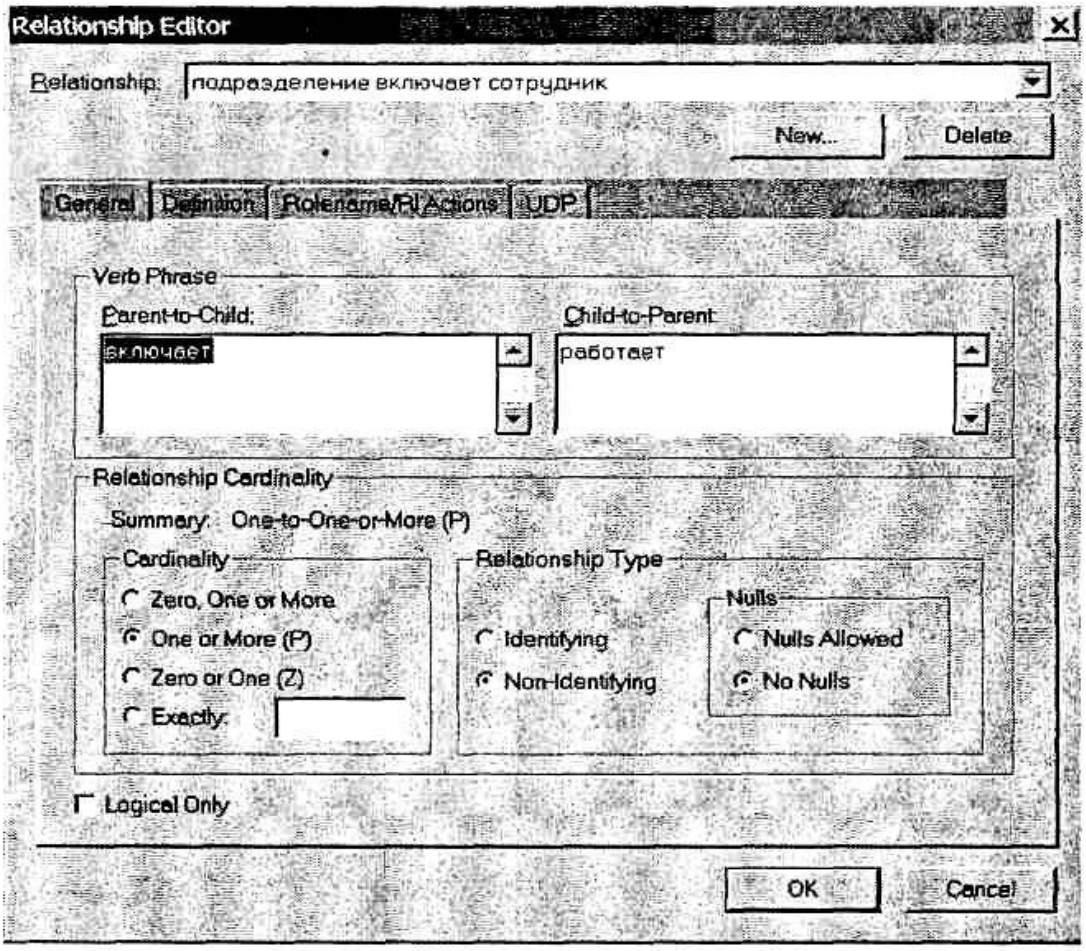


Рис. 2.92. Задание имен связи

Заданные при описании связи имена могут отображаться, а могут и не отображаться на экране. Для управления этим процессом можно воспользоваться позицией **Display Options/Relationships** (рис. 2.93).

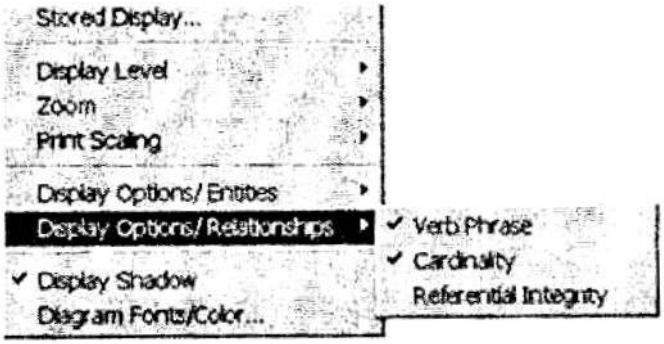


Рис. 2.93. Опции отображения связи (выбрана позиция Verb Phrase)

На рис. 2.94 представлен фрагмент ER-модели с отображенными на ней именами связей.

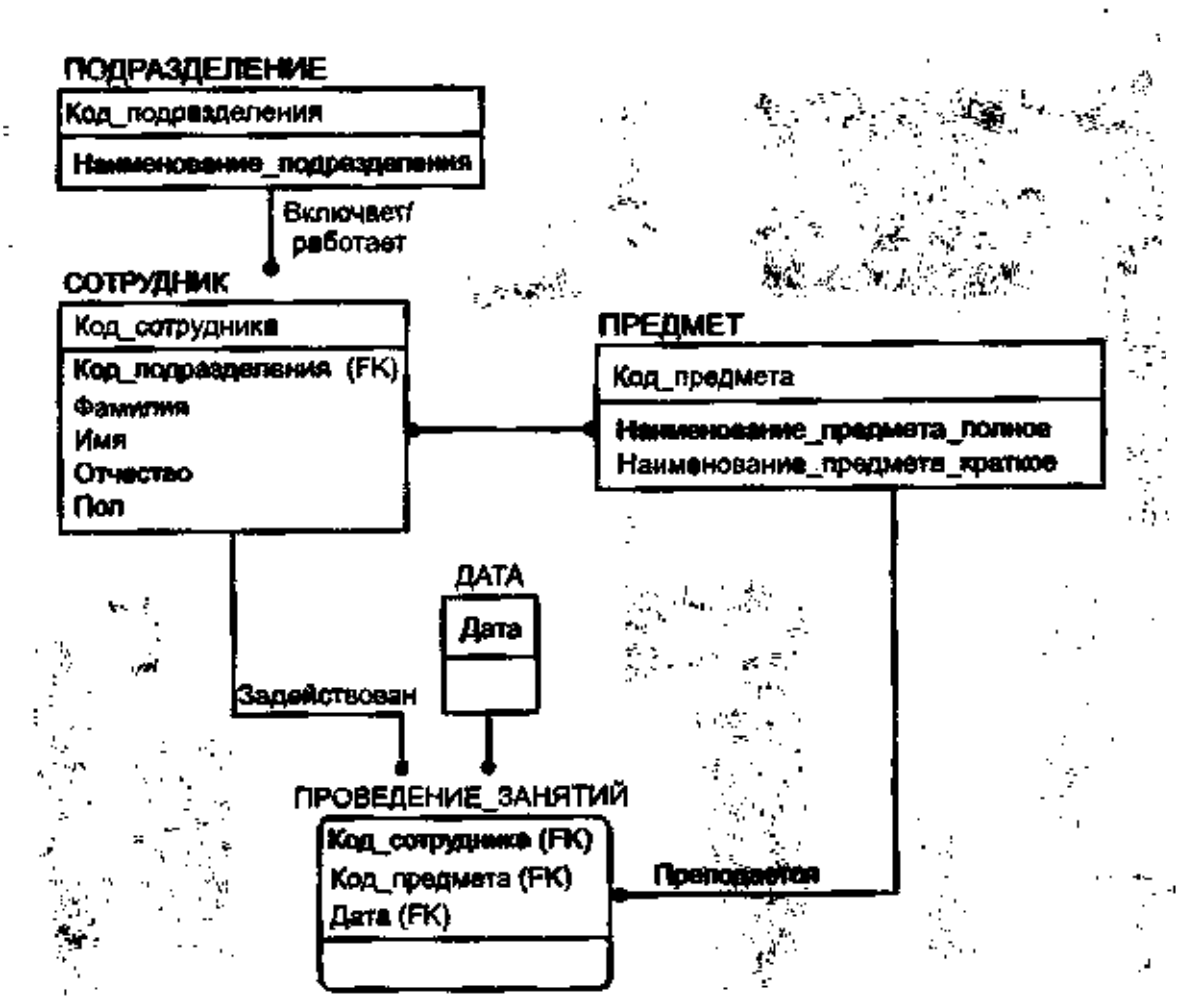


Рис. 2.94. Отображение имени связи

Задание нескольких связей между парой сущностей. В ERWin возможно задание нескольких связей между парой объектов (рис. 2.95). По умолчанию в этом случае на схеме в подчиненной сущности появляется только один внешний ключ.

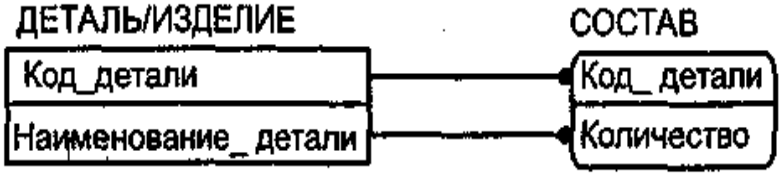


Рис. 2.95. Задание двух связей между сущностями (вариант по умолчанию)

Для того чтобы появились оба ключа, при описании связи на вкладке **Rolename/RI Action** (рис. 2.96) следует указать ролевые имена **(Rolename).**

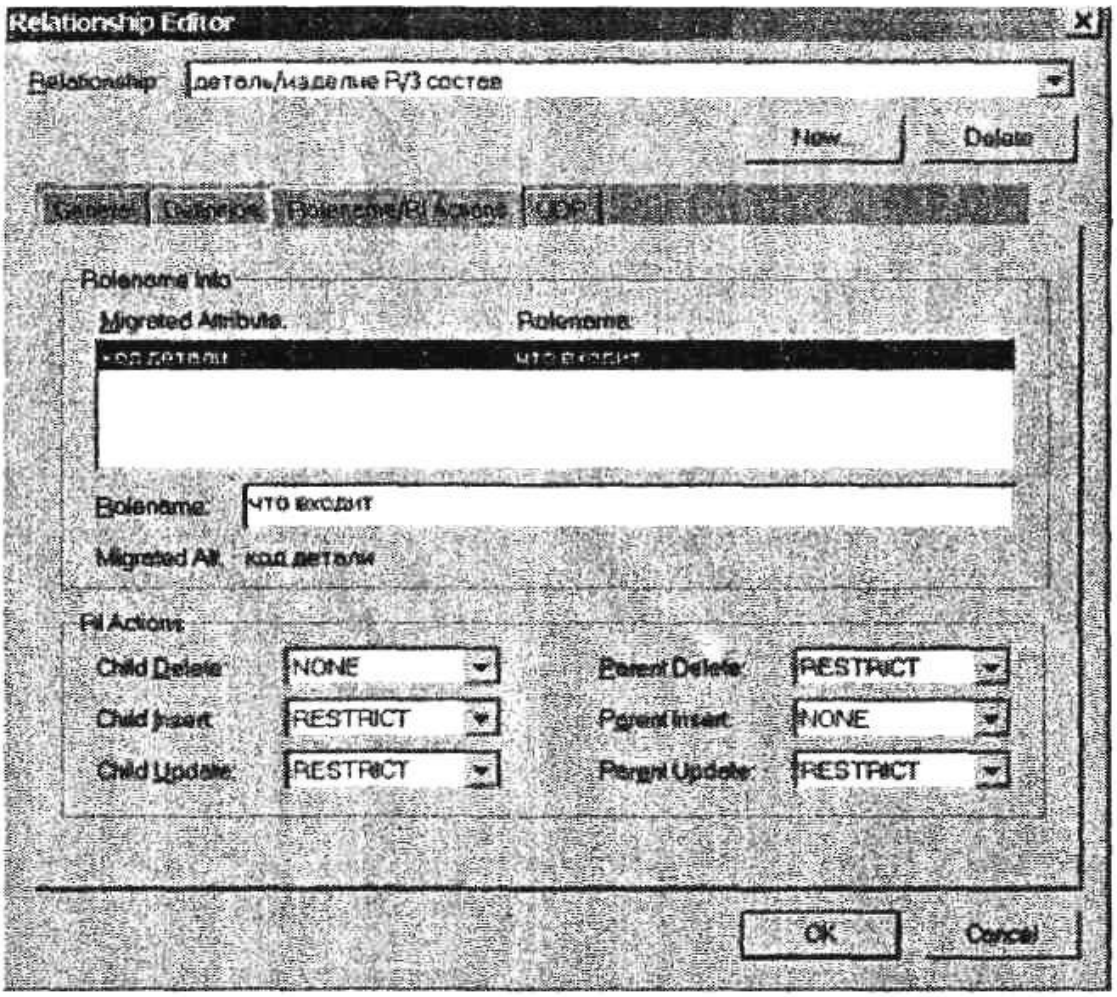


Рис. 2.96. Задание ролевого имени связи

На рис. 2.97 представлен фрагмент модели после задания ролевых имен и с опциями отображения, отмеченными на рис. 2.98.

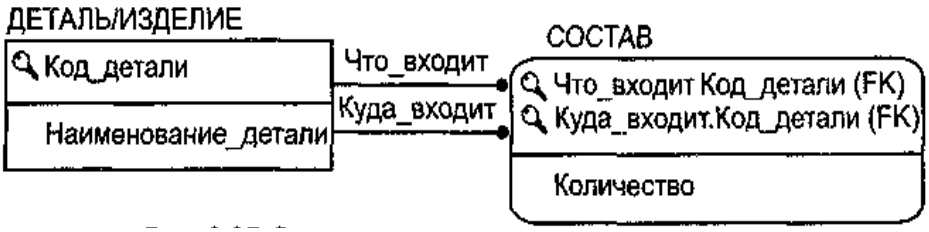


Рис. 2.97. Задание двух связей между сущностями
(с указанием ролевых имен)

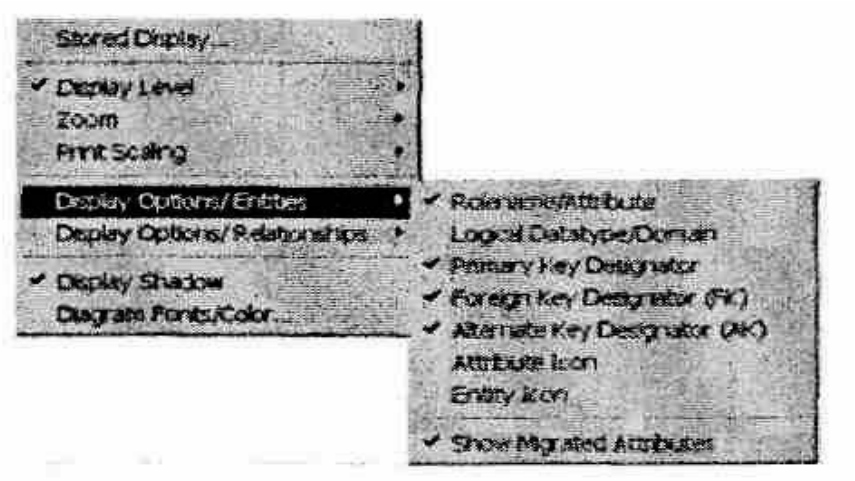


Рис. 2.98. Опции просмотра сущности

Вид модели в нотации IE. Модель в нотации IDEF (рис. 2.99) в нотации IE имеет вид, представленный на рис. 2.100.

Идентифицирующая связь, так же как и в нотации IDEF1X, изображается в нотации IE сплошной линией, а неидентифицирующая - пунктирной.

Соответствие между условными обозначениями нотаций IDEF1X и IE отражено в табл. 2.4.

Уровни отображения логической модели. Модель можно отображать на экране с разной степенью детализации: только сущности (Entity), все атрибуты (Attribute), сущности и их первичные ключи (Primary Key), описания (Definition), иконки (Icon). Для выбора нужного уровня отображения можно воспользоваться контекстным меню (рис. 2.101).

Во всех предшествующих примерах при изображении модели использовался уровень Attribute. На рис. 2.102 показан вид модели при выборе уровня Entity.

При выборе уровня Definition внутри прямоугольника, отображающего сущность, пишется то определение, которое было в окне **Definition** при описании сущности (рис. 2.103).

При использовании уровня иконок в прямоугольнике, соответствующем сущности, кроме названия сущности изображается специально подобранная иконка (рис. 2.104), что может повысить наглядность модели.

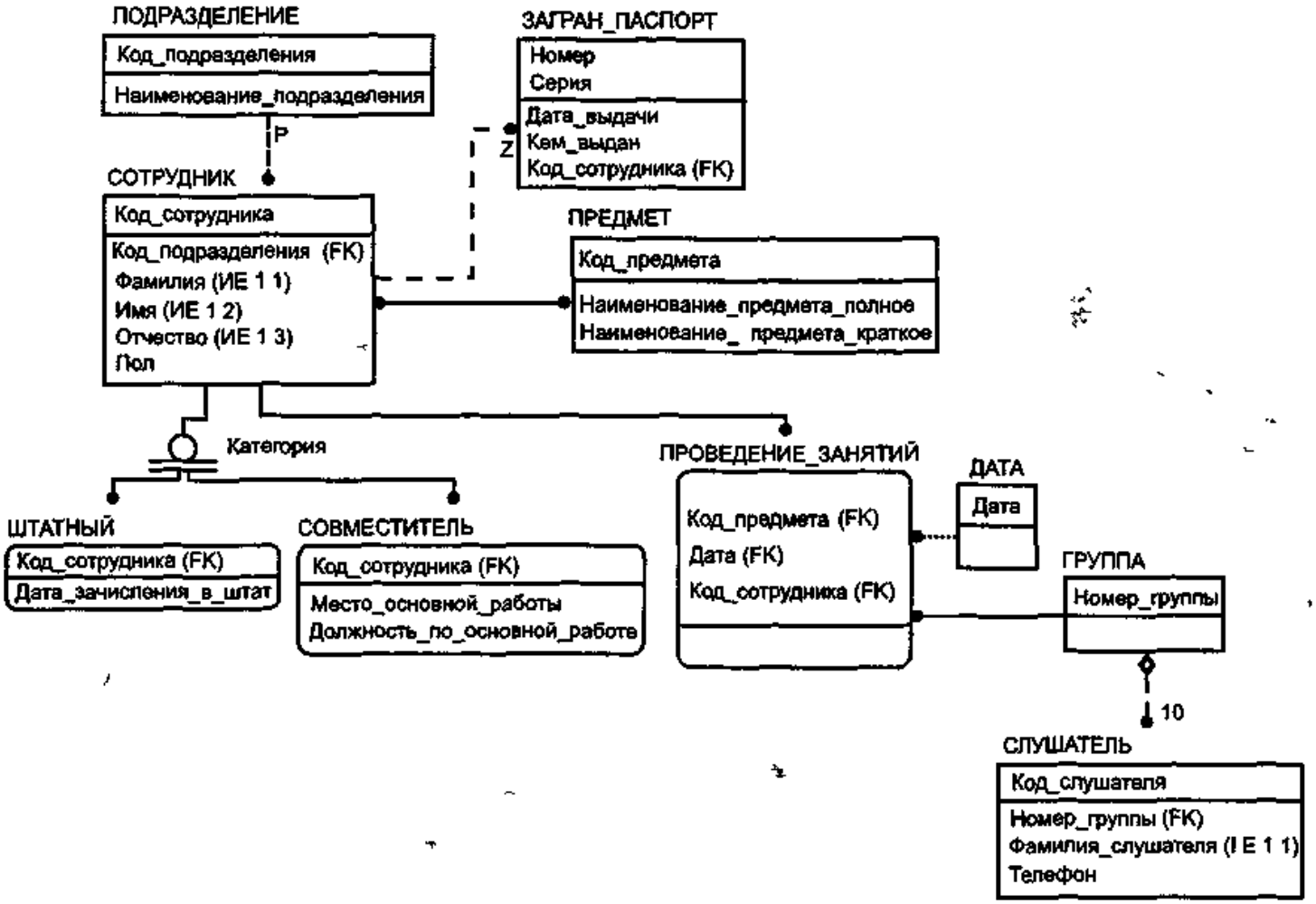


Рис. 2.99. ER-модель в нотации IDEF

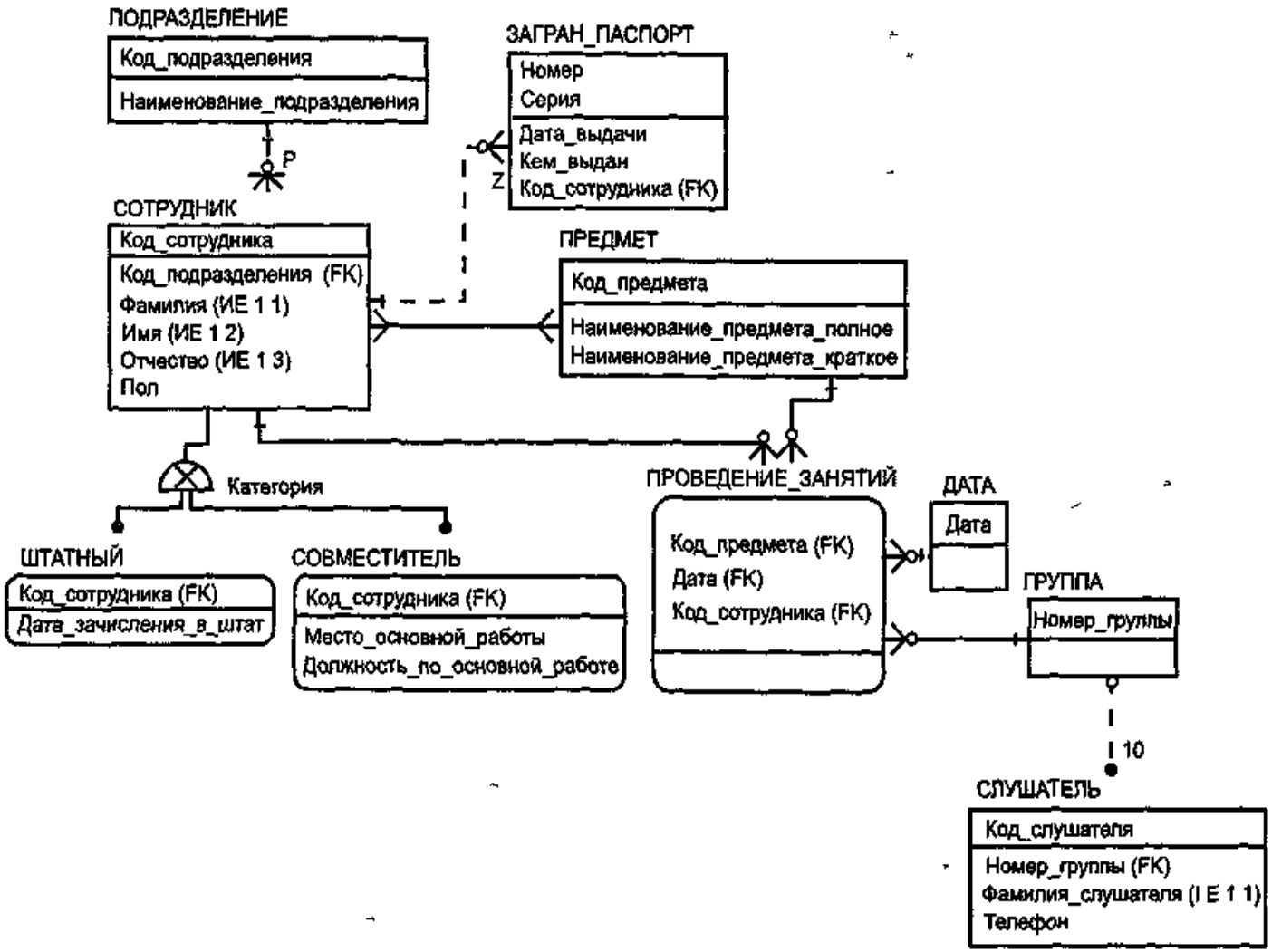
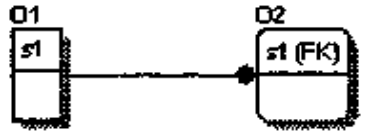
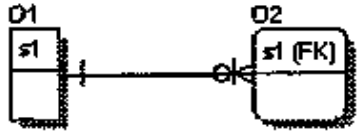
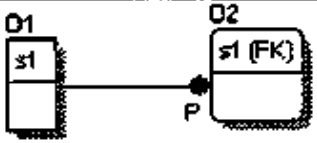
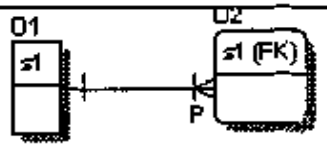
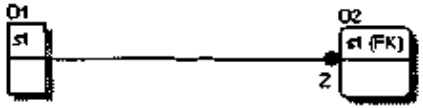
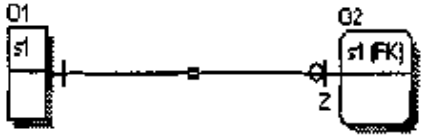
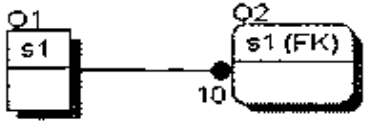
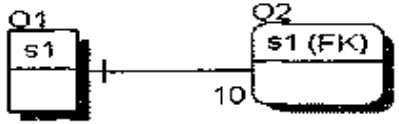
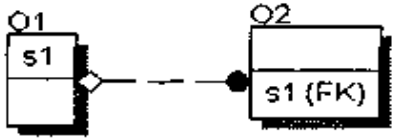
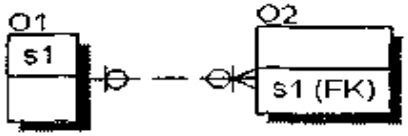
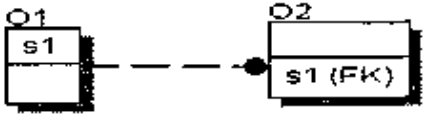
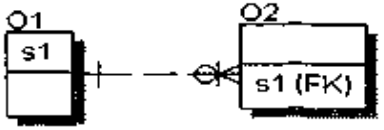


Рис. 2.100. ER-модель в нотации IE

Таблица 2.4

Вид связи	Изображение в IDEF1X	Изображение в IE
Идентифицирующая связь Zero, One or Many		
Идентифицирующая связь One or Many		
Идентифицирующая связь Zero, One		
Идентифицирующая связь Exactly		
Неидентифицирующая связь Zero, One or Many, Nulls allowed		
Неидентифицирующая связь Zero, One or Many, No nulls		

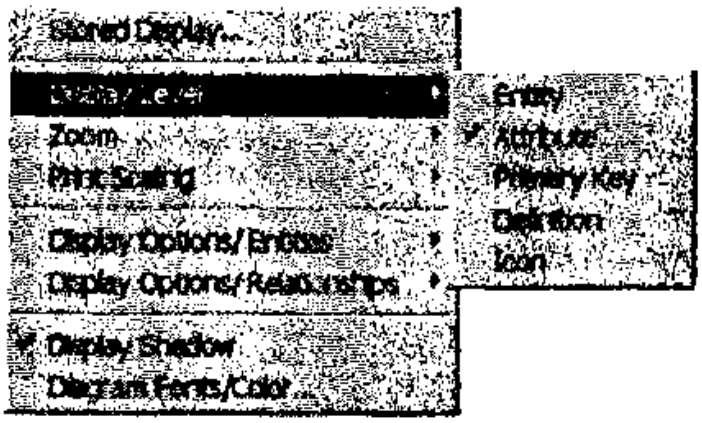


Рис. 2.101. Выбор уровня отображения модели (Display Level)

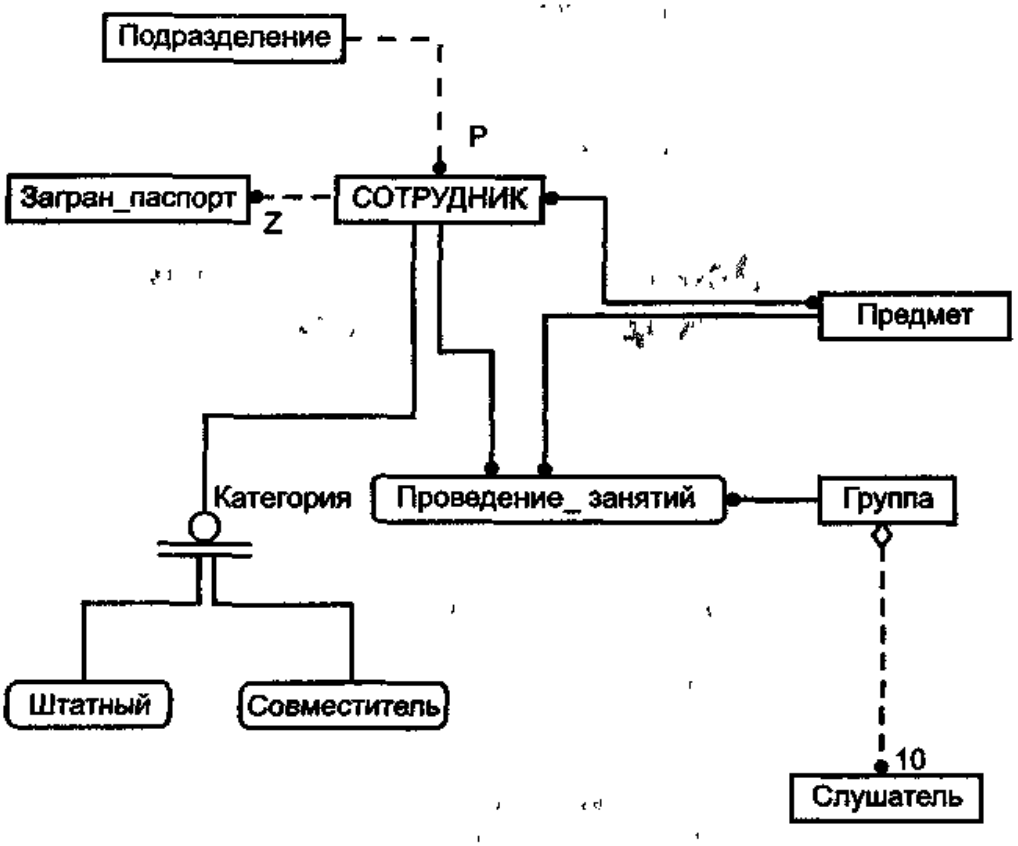


Рис. 2.102. Уровень сущности

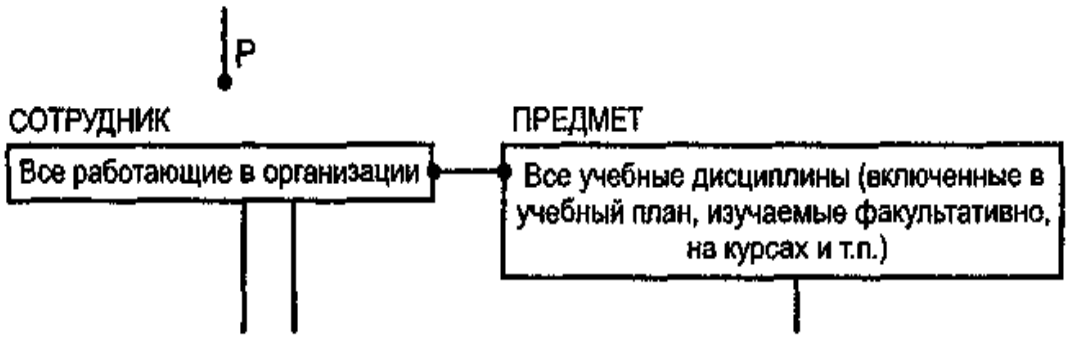


Рис. 2.103. Уровень описаний

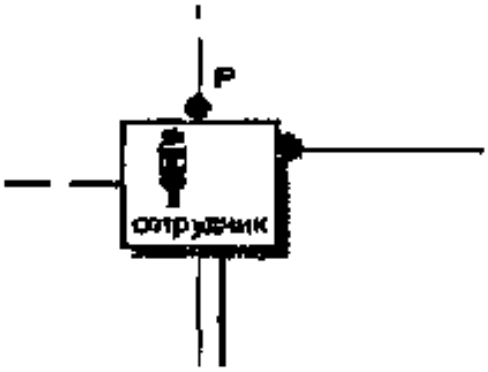


Рис. 2.104. Уровень иконок

2.6.3. Особенности методологии моделирования

Несмотря на имеющиеся различия в нотациях и реализации, моделирование в Design/IDEF и ERWin имеет много общего. Поэтому при построении концептуальной модели в ERWin за основу следует взять методологию моделирования, изложенную в разд. 2.4.2. В эту методологию должны быть внесены некоторые коррективы. Особенности в методологии при проектировании в среде ERWin и причины, обусловливающие их, изложены ниже.

1. При преобразовании из логической модели в физическую система ERWin автоматически создает связующую таблицу. Поэтому отпадает необходимость на уровне логического моделирования в преобразовании неспецифических связей в специфические.

2. В ERWin есть возможность указывать свойство «только физическая/только логическая» для сущности в целом и любого из ее атрибутов. Эту возможность можно использовать в процессе проектирования следующим образом:

- после построения общей концептуальной модели, включающей всю информацию, представляющую интерес для данной предметной области, определить состав информации, хранимой в базе данных;
- отметить те сущности/атрибуты, которые не требуется хранить в базе данных, признаком **Logical Only,** чтобы они не попадали в физическую модель.

Наличие указанных возможностей позволяет проектировщику иметь одну концептуальную схему и для этапа обсуждения со всеми заинтересованными лицами на всех стадиях моделирования, и для этапа генерации схемы базы данных.

В ERWin имеется целый ряд возможностей, которые отсутствуют в Design/IDEF.

В частности, в ERWin можно задавать «view». Представление создается в ERWin при описании физической модели, хотя фактически является логическим описанием БД с точки зрения конкретной задачи, т.е. относится к

этапу проектирования подсхем. Для выявления необходимости создания подсхем следует проанализировать запросы к БД.

Задание ограничений целостности является неотъемлемым элементом создания концептуальной модели. ERWin позволяет описывать ограничения целостности при описании атрибутов, а также при описании связей между объектами. Более подробно задание ограничений целостности изложено в главе 4.

На это следует обратить внимание

- Для построения ER-модели, которая корректно будет преобразована в модель целевой БД, при использовании CASE-средств необходимо не только хорошо понимать отображаемую предметную область и уметь адекватно описывать ее с использованием языковых возможностей данной системы, но и хорошо представлять себе алгоритмы преобразования, заложенные в используемой системе.
- Концептуальное проектирование является первым и важнейшим этапом проектирования структурированных баз данных.
- Концептуальная модель используется разными категориями проектировщиков ИС и заказчиков. Поэтому и те, и другие должны понимать основы концептуального моделирования.
- В настоящее время имеется достаточно большое число CASE-систем, позволяющих автоматизировать процесс построения ER-моделей. При этом следует иметь в виду, что ни одна из систем в принципе не может обеспечить проверку правильности построения ER-модели относительно ее соответствия моделируемой предметной области.
- CASE-средства не только позволяют автоматизировать процесс проектирования ИС, но и облегчают организацию коллективной разработки проекта, что еще больше повышает значимость их использования при создании больших и сложных проектов.
- CASE-средства могут быть использованы не только при первоначальном проектировании систем, но и перепроектировании уже существующих систем в процессе их развития.

Контрольные вопросы

1. Что называется предметной областью?
2. Что называется концептуальной моделью? Для каких целей она служит?
3. Перечислите основные компоненты концептуальной модели.
4. Какие требования предъявляются к концептуальной модели?
5. Какие преимущества дает использование ER-моделирования при создании БД?
6. Что называется классом объектов?

7. Какие разновидности объектов выделяются в базовой ER-модели? Какие графические обозначения используются для изображения каждого вида объектов?

8. Приведите примеры из любых предметных областей для каждой разновидности объектов.

9. Какие разновидности свойств объектов выделяются в базовой ER-модели? Какие графические обозначения используются для изображения каждого вида свойств?

10. Приведите примеры из любых предметных областей для каждой разновидности свойств.

11. Что называется зависимыми от идентификации сущностями?

12. Что следует выделять в качестве самостоятельного объекта в ER-модели?

13. Какие интегральные характеристики класса объектов обычно фиксируются при описании предметной области? Как они используются при проектировании БД?

14. В каких случаях в концептуальной модели следует в явном виде отображать класс объектов?

15. Какие разновидности связи между объектами выделяются в базовой ER-модели? Какие графические обозначения используются для изображения каждого вида связи?

16. Приведите примеры из любых предметных областей для каждой разновидности связи.

17. В каком случае следует вводить в модель обобщенный объект?

18. Какую информацию о предметной области дает граф пересечений?

19. Какие CASE-средства вы знаете?

20. Чем отличаются известные вам методологии ER-моделирования друг от друга?

21. Какие разновидности объектов выделяются в ER-модели, построенной в нотации IDEF1X?

22. В чем общность и различие процесса ER-моделирования в CASE-системах Design/IDEF и ERWin?

23. Что называется дискриминатором? Для каких целей он служит?

24. Что называется инверсным входом (Inversion Entry)? В каких случаях при описании атрибута следует использовать этот признак?

25. В каких случаях в ERWin следует использовать при описании атрибута признак Logical Only (только логический)?

26. В чем отличие полных категорий (Complete sub-category) от неполных (Incomplete sub-category)?

Глава 3 ДАТАЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

3.1. Общие сведения о даталогическом проектировании

3.1.1. Исходные данные для даталогического проектирования

Подходы к проектированию логической структуры БД существенно зависят от типа модели данных. Рассмотрим вопросы даталогического проектирования применительно к структурированным моделям данных.

В разд. 1.5 было дано понятие даталогического проектирования и определен состав работ на этой стадии (*см.* рис. 1.23). Рассмотрим эту схему подробнее.

Любая СУБД оперирует с допустимыми для нее логическими единицами данных, а также разрешает использование определенных правил композиции логических структур более высокого уровня из составляющих информационных единиц более низкого уровня. Кроме того, многие СУБД накладывают количественные и иные ограничения на структуру базы данных. Поэтому, прежде чем приступить к построению даталогической модели, необходимо детально изучить особенности СУБД, определить факторы, влияющие на выбор проектного решения, ознакомиться с существующими методиками проектирования, а также провести анализ имеющихся средств автоматизации проектирования, возможности и целесообразности их использования.

Хотя даталогическое проектирование соотносится с логической структурой базы данных, на него оказывают влияние возможности физической организации данных, предоставляемые конкретной СУБД. Поэтому знание особенностей физической организации данных является полезным при проектировании логической структуры.

Логическая структура базы данных, а также сама заполненная данными база данных являются отображением реальной предметной области. Поэтому на выбор проектных решений самое непосредственное влияние оказывает специфика отображаемой предметной области, отраженная в инфологической модели.

Все шаги проектирования даталогической модели выполняются итеративно. Причем вероятны итерации не только внутри стадии даталогического проектирования, но и с «захватом» других стадий проектирования БД.

3.1.2. Результат даталогического проектирования

Конечным результатом даталогического проектирования является описание логической структуры базы данных на ЯОД. Однако если проектирование выполняется вручную, то для большей наглядности сначала строится схематическое графическое изображение структуры базы данных. При этом должно быть обеспечено однозначное соответствие между конструкциями языка описания данных и графическими обозначениями информационных

единиц и связей между ними. Графическое представление используется и при автоматизированном проектировании структуры базы данных как интерфейсное средство общения с проектировщиком и при документировании проекта.

Спроектировать логическую структуру базы данных - означает определить все информационные единицы и связи между ними, задать их имена; если для информационных единиц возможно использование разных типов, то определить их тип. Следует также задать некоторые количественные характеристики, например длину поля.

3.1.3. Подход к даталогическому проектированию

Каждому типу модели данных и каждой разновидности модели, поддерживаемой конкретной СУБД, присущи свои специфические особенности. Вместе с тем имеется много общего во всех структурированных моделях данных и принципах проектирования БД в их среде. Все это дает возможность использовать единый методологический подход к проектированию структуры базы данных.

В БД отражается определенная предметная область. Поэтому процесс проектирования БД предусматривает предварительную классификацию объектов предметной области, систематизированное представление информации об объектах и связях между ними.

На проектные решения оказывают влияние особенности требуемой обработки данных. Поэтому соответствующая информация должна быть определенным образом представлена и проанализирована на начальных этапах проектирования БД.

Данные о предметной области и особенностях обработки информации в ней фиксируются в инфологической модели. В такой модели должна быть отображена вся информация, циркулирующая в информационной системе, но это вовсе не означает, что вся она должна храниться в базе данных. В связи с этим одним из первых шагов проектирования является определение состава БД, т.е. перечня тех показателей, которые целесообразно хранить в БД.

При проектировании логической структуры БД осуществляются преобразование исходной инфологической модели в модель данных, поддерживаемую конкретной СУБД, и проверка адекватности полученной даталогической модели отображаемой предметной области.

Для любой предметной области существует множество вариантов проектных решений ее отображения в даталогической модели. Методика проектирования должна обеспечивать выбор наиболее подходящего проектного решения.

Минимальная логическая единица данных (несмотря на их разные названия) семантически для всех СУБД одинакова и соответствует либо идентификатору объекта, либо свойству объекта или процесса.

Связи между сущностями предметной области, отраженные в инфологической модели, могут отображаться в даталогической модели либо

посредством совместного расположения соответствующих им информационных элементов, либо путем объявления связи между ними. Связь может передаваться как на внутризаписном, так и на межзаписном уровне.

Не все виды связей, существующие в предметной области, могут быть непосредственно отображены в конкретной даталогической модели. Так, многие СУБД не поддерживают непосредственно отношение M:M между элементами. В этом случае в даталогическую модель вводится дополнительный вспомогательный элемент, отображающий эту связь (таким образом, отношение M:M как бы разбивается на два отношения 1:M между этим вновь введенным элементом и исходными элементами).

Следует обратить внимание на то, что отношения, имеющие место в предметной области и отражаемые в ИЛМ, могут быть переданы не только посредством структуры базы данных, но и программным путем (т.е. всегда существует альтернатива между декларативным и процедурным способом описания явления). Например, при отображении обобщенных объектов можно не выделять подклассы на уровне логической структуры базы данных. В этом случае подклассы будут выделяться программным путем при обработке хранимых данных.

Решение о том, какой из способов отображения (структурный/декларативный или программный/процедурный) следует использовать в каждом конкретном случае, будет зависеть от многих факторов, таких, как стабильность отображаемой сущности, объем номенклатуры, особенности СУБД, характер обработки данных и др. Так, если в предметной области используется классификация сотрудников по полу, в базе данных не следует создавать классификатор полов, поскольку он будет содержать всего две позиции и никогда не меняется. Как правило, не следует выделять по этому признаку и соответствующие подклассы для объектов предметной области, потому что в большинстве случаев они обрабатываются совместно и в основном имеют одинаковый набор свойств, характеризующий их. Однако в некоторых предметных областях деление на такие подклассы может быть целесообразным.

Если же отображаемая сущность не стабильна, то ее лучше передавать посредством данных, так как в противном случае будет часто требоваться преобразование программы, что обычно обеспечить труднее, чем изменение данных.

При отображении обобщенных объектов в БД возможны разные варианты: хранить информацию обо всем обобщенном объекте в одном файле/таблице, каждому подклассу объектов низшего уровня выделять отдельные самостоятельные файлы/таблицы. Оба эти варианта могут быть использованы в любой СУБД. В первом случае подчеркивается общность объектов разных подклассов, входящих в обобщенный объект. Во втором случае, напротив, обобщенный объект как единое целое не отображается в структуре базы данных.

Другие способы отображения связаны с явным или неявным выделением подклассов в логической структуре БД. Неявное выделение подкласса заключается в том, что в записи отводятся поля для фиксации значений

свойств, общих для объектов разных подклассов, и значения признака подкласса, а вместо полей, наличие которых зависит от подкласса, используется одно поле с переменным составом, содержание которого будет зависеть от того, к какому подклассу относится описываемый объект.

Реализация принципа явного выделения подклассов в структуре БД существенно зависит от специфики СУБД.

При проектировании логической структуры БД основное значение имеет специфика отображаемой предметной области. Однако, как отмечалось выше, и характер обработки информации оказывает влияние на принимаемое проектное решение. Например, рекомендуется хранить вместе информацию, часто обрабатываемую совместно, и, наоборот, разделять по разным файлам информацию, не использующуюся одновременно. Информацию, используемую часто, и информацию, частота обращения к которой мала, также следует хранить в разных файлах, причем последнюю может оказаться выгодным вынести в архивные файлы, а не поддерживать в составе БД.

Как отмечалось выше, в ИЛМ описывается не отдельный объект, а класс объектов. Но в редких случаях бывает, что класс включает в себя только один экземпляр объекта. Например, если предметной областью является какой-то конкретный институт, то класс объектов ИНСТИТУТ будет содержать только один экземпляр объекта. Таким «вырожденным» классам объектов обычно не ставится в соответствие отдельный файл базы данных.

3.1.4. Определение состава базы данных

При переходе от инфологической модели к даталогической следует иметь в виду, что инфологическая модель включает в себя всю информацию о предметной области, необходимую и достаточную для проектирования БД. Это не означает, что все сущности, зафиксированные в ИЛМ, должны в явном виде отражаться в даталогической модели. Прежде чем строить даталогическую модель, необходимо решить, какая информация будет храниться в базе данных. Например, в инфологической модели должны быть отражены вычисляемые показатели, но вовсе не обязательно, что они должны храниться в базе данных.

Существуют разные подходы к определению состава показателей, которые должны храниться в базе данных. Согласно одному из них в БД должны храниться только исходные показатели, все производные показатели должны быть получены расчетным путем в момент реализации запроса (этот подход иногда называют принципом синтезирования, имея в виду возможность синтезировать требуемые показатели из хранимых в информационной базе данных).

Такой подход имеет очевидные достоинства:

1) простота и однозначность в принятии решения «что хранить»;

2)отсутствие неявного дублирования информации со всеми вытекающими из этого последствиями (меньше объем памяти, чем при хране-

нии как исходных, так и производных показателей; проще проблемы контроля целостности данных);

3) потенциальная возможность получить любой расчетный показатель, а не только те, которые хранятся в БД.

При принятии решения о хранении расчетных показателей должны учитываться разные факторы.

Рассмотрим некоторую гипотетическую предметную область, представляющую собой учебное заведение. Учащимся этого заведения начисляется стипендия. Существует четкий алгоритм начисления стипендии. Предположим, что он выглядит следующим образом.

1. Стипендия начисляется только тем, кто успешно сдал все экзамены в сессию (т.е. сдал в срок и не получал неудовлетворительных оценок), а также учащимся, имеющим детей.

2. Известен размер обычной стипендии.

3. Тем, кто не имеет удовлетворительных оценок, назначается стипендия на 25% выше, чем обычная.

4. Тем, кто имеет только отличные оценки, стипендия увеличивается на 50% от размера обычной стипендии.

Предположим также, что размер стипендии не может изменяться в течение семестра. Таким образом, размер стипендии является вычисляемым показателем и может не храниться в базе данных, а каждый раз определяться расчетным путем. Однако в рассматриваемой ситуации его все-таки лучше хранить в БД по следующим основным причинам:

а) полученная величина многократно используется в дальнейшем;

б) алгоритм определения размера стипендии имеет достаточно сложную логику, и для выполнения расчета требуется просмотреть несколько файлов (некоторые из них являются большими по объему, что также замедляет процесс обработки);

в) размер стипендии не должен изменяться в течение семестра. Кроме того, имея реальный файл «Начисленная стипендия», легче контролировать его полноту и достоверность.

Предположим также, что к БД достаточно часто обращаются с запросом о среднем балле какого-либо студента или среднем балле по той или иной дисциплине и т.п.

При изменении любой оценки или получении новой средний балл меняется, и если вы вдруг решите хранить и среднюю величину, и исходные оценки, то обеспечение их соответствия может стать проблемой. При правильной организации файлов вычисление средней величины во время исполнения запроса не представляет проблемы ни с точки зрения времени обработки, ни в аспекте задания запроса. Поэтому хранить средний балл не следует.

3.1.5. Введение искусственных идентификаторов

При отображении объекта в файл базы данных идентификатор объекта будет являться полем этого файла, причем в большинстве случаев - ключевым полем, т.е. полем, однозначно идентифицирующим запись. Однако в некоторых случаях появляется необходимость введения искусственных идентификаторов, или, как их еще называют, кодов. Такими случаями являются следующие.

1. В предметной области может наблюдаться синонимия, т.е. может случиться, что естественный идентификатор объекта не обладает свойством уникальности. Например, среди сотрудников предприятия могут быть однофамильцы. В этом случае для обеспечения однозначной идентификации объектов предметной области в информационной системе целесообразно использовать искусственные коды.

2. Если объект участвует во многих связях/процессах, то для их отображения создается несколько файлов/таблиц, в каждом из которых повторяется идентификатор объекта. Для того чтобы не использовать во всех файлах длинный естественный идентификатор объекта, можно ввести и использовать более короткий код. Например, вместо длинных наименований продукции обычно используются их кодовые обозначения. Это не только сэкономит память, но и сократит трудоемкость ввода информации (хотя последнего можно достичь и другим путем).

3. Если естественный идентификатор может изменяться со временем (например, фамилия), то это может вызвать много проблем, если наряду с таким динамическим идентификатором не использовать статический искусственный идентификатор.

Когда присваиваются идентификаторы каким-либо объектам, то желательно, чтобы эти идентификаторы были постоянными. Например, в учебных заведениях всегда каким-либо образом обозначаются учебные группы. Для этого можно, например, использовать номер курса, на котором в настоящее время учится данная группа, и какие-то другие символы (подобно тому, как обозначаются классы в школе). Но в этом случае идентификатор группы будет каждый год меняться. А можно для обозначения группы использовать, например, две последние цифры года поступления. В этом случае идентификатор группы будет постоянным. Использование постоянных идентификаторов снимает множество проблем при функционировании информационной системы.

Встречаются и иные случаи, когда целесообразно вводить искусственные идентификаторы. Некоторые из них будут рассмотрены далее при обсуждении проблем проектирования БД.

3.1.6. Критерии оценки БД

Для оценки проектируемой/спроектированной БД может быть использовано множество критериев. Значимость этих критериев, в свою очередь, бу-

дет зависеть от большого числа разнообразных факторов. Прежде всего эта значимость зависит от самого критерия: есть критерии, значимость которых не только никогда не понижается, а, наоборот, постоянно растет (это такие критерии, как адекватное отображение действительности, удовлетворение разнообразных потребностей пользователей и т.п.). Значимость других критериев становится менее существенной с ростом возможностей техники и программного обеспечения (например, занимаемый базой данных объем памяти).

На значимость критерия оценки влияют особенности ИС, для которой создается проект (например, для систем, работающих в реальном масштабе времени, очень важна скорость реакции системы на запросы, для банковских систем - защита информации и надежность системы).

Все критерии оценки являются взаимосвязанными и часто противоречивыми: улучшение показателей по одному из критериев может привести к ухудшению значений показателей, оценивающих модель по другим критериям. Необходима комплексная оценка проекта по всей совокупности критериев, которые могут быть как количественными, так и качественными.

Перечислим основные критерии, используемые при оценке БД.

1. *Адекватность* - соответствие базы данных реальной предметной области. Естественно, что БД не должна искажать предметную область. Это является важнейшим требованием к БД, без соблюдения которого бессмысленно соблюдение всех остальных требований. Оценка адекватности является не только очень важной, но и очень трудной задачей, поскольку некоторые несоответствия трудно выявить. Адекватность должна быть обеспечена как на уровне структуры БД, так и при задании ограничений целостности. Так, например, если в предметной области могут быть однофамильцы, а вы зададите ФИО как ключ, то запись, касающаяся однофамильца, не сможет быть введена в базу данных.

2. *Полнота* - возможность удовлетворения существующих и новых потребностей пользователей. Использование подхода «от предметной области» к проектированию баз данных и сама идеология банков данных как интегрированного взаимоувязанного хранилища данных способствуют обеспечению этого критерия. Хранение в БД детализированных показателей также повышает возможности удовлетворения разнообразных (в том числе и нерегламентированных) потребностей пользователей. Полнота является одним из проявлений адекватности БД.

3. *Адаптируемость.*

3.1. *Адаптируемость к изменениям в предметной области.*

3.1.1. *Устойчивость схемы базы данных* - отсутствие необходимости в изменении структуры БД при изменении предметной области. В теории БнД широко используется понятие независимости программ от данных и данных от программ. Не менее, а даже более значимой является проблема обеспечения независимости логической модели БД от изменений, происходящих в предметной области. Устойчивость модели является лучшим проявлением свойства адаптируемости системы. Обеспечение устойчивости модели базы

данных к изменениям предметной области фактически снимает проблему независимости программ от данных, так как в этом случае структуры данных меняться не будут.

Поясним суть данного показателя на примере. Предположим, необходимо хранить информацию об успеваемости студентов. Ниже приведены некоторые из возможных вариантов структуры БД для хранения этой информации. Вариант 1 предполагает, что создается таблица. В ее строках помещаются фамилии студентов, графами являются названия предметов, а значениями соответствующих полей будет оценка, полученная данным студентом по данному предмету. Каждому студенту соответствует одна запись в БД. Такой вариант очень похож на «Книги баллов», которые традиционно ведутся в деканатах. Для обсуждения проблемы устойчивости модели несущественно, а для оценки БД по другим критериям будет иметь значение, сколько таблиц создано (так как студенты учатся по разным учебным планам, то при создании только одной таблицы будет много пустых значений полей, содержащих оценки, а каждая запись будет содержать слишком большое число полей).

Вариант 1

ФИО	Высш_мат.	Ин._яз.	БД	…
Иванов	5	4	5	…
Якушкина	4	5	4	…

В варианте 2 таблица содержит только три поля: «ФИО», «Предмет», «Оценка». Для каждого студента будет создано столько записей, сколько экзаменов сдал этот студент.

Вариант 2

ФИО	Предмет	Оценка
Иванов	Высш. мат	5
Якушкина	Высш. мат	4
Иванов	Ин. яз.	4
Якушкина	Ин. яз	5
…	…	…

Если будет выбран вариант 1, то при каждом изменении в учебном плане необходимо менять структуру базы данных. Во втором случае никакие изменения структуры БД не потребуются. Более того, при создании БД по варианту 2 вообще не нужно знать действующие учебные планы. Решение будет универсальным и может использоваться в любом учебном заведении без необходимости «подстройки» на конкретное учебное заведение. Кстати, *универсальность* также может использоваться как критерий оценки БД.

Приведем другой пример, показывающий влияние выбранного типа данных на устойчивость спроектированной БД. Предположим, что в пред-

метной области для кодирования какой-либо номенклатуры используется цифровой код и в базе данных для соответствующего поля был выбран числовой тип данных. Если возникнет необходимость перейти на буквенную или буквенно-числовую систему кодирования, то в БД придется менять тип данных у соответствующего поля. Если бы тип данных при проектировании БД изначально был определен как текстовой, то изменения бы не потребовались.

С рассматриваемым критерием будет тесно связан критерий *затрат на поддержание системы в работоспособном состоянии*. С затратами на адаптацию структуры БД будут непосредственно связаны и *затраты на адаптацию прикладного программного обеспечения*.

3.1.2. *Простота и эффективность внесения изменений*. Речь может идти как об изменении структуры базы данных в случае возникновения такой необходимости, так и об обычной корректировке значений данных в базе данных.

3.1.2.1. *Простота корректировки структуры БД данных*. Например, некоторые типы полей трудно преобразовать в другие. Особенно внимательными нужно быть при определении полей связей, так как их изменение повлечет за собой целую цепочку изменений.

3.1.2.2. *Простота и трудоемкость корректировки значений данных*. Прежде всего следует обратить внимание на аномалии, возникающие при корректировке ненормализованных структур.

Одним из показателей простоты корректировки может быть число изменений, которые необходимо провести при наступлении одного события в предметной области.

Например, предположим, что имеется база данных, содержащая сведения о сотрудниках учебного заведения, и в ней фиксируются некоторые биографические и прочие справочные данные, а также информация о том, какие учебные предметы может вести каждый преподаватель. Преподаватель может владеть несколькими предметами. Сравним следующие три проектных решения:

Проектное решение 1:

Ф1(ФИО, дата_рождения, пол, ..., телефон, название_предмета).

Проектное решение 2:

Ф1(ФИО, дата_рождения, пол, ...,);

Ф2 (ФИО, название_предмета).

Проектное решение 3:

Ф1(код_сотрудника, ФИО, дата_рождения, пол, ...,);

Ф2 (код_сотрудника, название_предмета).

Следует сразу отметить, что варианты 1 и 2 являются неудовлетворительными и использованы только для иллюстрации недостатков, связанных с подобными решениями.

В первом случае имеем таблицу, находящуюся в первой нормальной форме (1НФ). Если преподаватель владеет несколькими предметами, то в таблице Ф1 ему будет соответствовать несколько строк.

Смена фамилии каким-либо сотрудником приведет в варианте 1 к корректировке стольких записей, сколько предметов ведет данный преподаватель, в варианте 2 - еще на одну запись больше, а в варианте 3 - к корректировке только одного значения. Изменение же номера телефона приведет в варианте 1 также к корректировке стольких записей, сколько предметов ведет данный преподаватель, а в вариантах 2 и 3 - к корректировке только одной записи.

Проектное решение 1 - ненормализованная структура. Она плоха тем, что приводит к большому дублированию информации. Варианты 1 и 2 - оба представляют нормализованную структуру и отличаются тем, что в последнем варианте введен искусственный идентификатор. Именно вариант 3 является наиболее предпочтительным.

Если «вдруг» (вообще-то этого лучше не делать) в базе данных в качестве поля связи определено поле, которое может менять свое значение, то следует обратить внимание на возможность автоматической корректировки связанных полей при соответствующем задании правил обеспечения ограничений целостности связи.

3.2. *Адаптация к изменениям информационных потребностей пользователей, возможность удовлетворения нерегламентированных запросов.* Например, если хранить в БД детальные данные, то любые производные данные можно получить при возникновении необходимости в них; если же хранить только какие-либо сводные данные, но не хранить исходные, то получить информацию, отличную от хранимой, в большинстве случаев нельзя.

3.3. *Адаптация к изменениям используемых программных и технических средств.* Основным способом обеспечения этого требования является соблюдение стандартов, а также, по возможности, использование при выборе проектного решения таких средств, которые являются широко распространенными, а не специфическими для конкретной системы. Так, например, использование экзотических типов полей скорее всего приведет к проблемам при переносе системы в другую среду или при обработке информации в гетерогенной среде.

Одним из проявлений рассматриваемого свойства является масштабируемость. Ведущие разработчики программных продуктов уделяют большое внимание обеспечению этих свойств.

4. *Универсальность.* Может быть обеспечена разными способами, например реализацией возможности настройки системы на особенности предметной области, определенными приемами при проектировании структуры БД и программного обеспечения. Особое значение приобретает при создании отчуждаемых проектов, ориентированных на конечных пользователей, не являющихся специалистами по машинной обработке данных.

5. *Сложность структуры БД.* Речь может идти как о сложности самой поддерживаемой в данной СУБД модели данных, так и о сложности логической структуры конкретной спроектированной БД.

Сложность модели будет определяться числом разнообразных информационных единиц, допустимых в ней, способами их соподчинения и связыва-

ния, накладываемыми ограничениями. Самыми простыми из структурированных моделей БД являются реляционные.

Естественно, что показатели сложности спроектированной БД будут зависеть от типа поддерживаемой модели БД. Сравнение по этому показателю баз данных, спроектированных в среде разных СУБД, будет иметь свою специфику. Для реляционной модели сложность будет характеризоваться числом таблиц и полей в БД, числом индексных файлов (индексов). В принципе, чем меньше сложность БД, тем лучше. Однако снижение сложности наряду с положительными результатами часто приводит ко многим отрицательным последствиям. Так, для реляционных систем самой «простой» БД будет одно универсальное отношение, но к каким последствиям приведет использование такого проектного решения, хорошо известно из теории нормализации.

Критерий «сложность» никогда не рассматривается как самостоятельный.

6. *Степень дублирования данных в БД.* Различают необходимое, контролируемое и неконтролируемое дублирование. Но какими бы причинами ни было вызвано дублирование данных, оно всегда ведет к необходимости поддержки идентичности всех копий дублируемых значений, росту требуемого объема памяти, повышению трудоемкости корректировки, увеличению числа полей в БД, что повышает ее сложность.

7. *Сложность последующей обработки.* Оценить этот показатель достаточно трудно, поскольку его значение зависит как от предполагаемой обработки, так и от возможностей языка манипулирования данными конкретной СУБД. Тем не менее для большинства СУБД справедливыми являются следующие утверждения:

- легче обрабатывать один файл, чем несколько связанных файлов;
- легче объединить несколько полей, чем выделить отдельные составляющие из единого поля (например, из «Адреса» - страну, город и т.п., из «ФИО» - фамилию, имя, отчество и т.п.). Если, например, в каких-либо выходных документах необходимо вывести фамилию и инициалы, то в случае раздельного хранения полей это можно легко сделать, например, просто изменив шаблон вывода для полей «имя» и «отчество» на (X.)). Однако хранение каждого элемента составной единицы информации (СЕИ) в виде отдельных полей имеет и очевидные недостатки: база данных становится сложнее, затрачивается больше времени при создании файла на описание его структуры, увеличивается объем служебной информации и объем памяти, требуемой для хранения данных;
- обработка «неэлементарных» полей в реляционных системах всегда представляет сложность (это вызвано тем, что с точки зрения СУБД поле остается элементарной единицей). Например, если в БД «Кадры» включено поле «Дети», в котором в записи о сотруднике фиксируют сведения обо всех его детях, то задать запросы типа: «Выделить всех сотрудников, имеющих больше трех детей» или «Определить среднее число детей у сотрудников» будет достаточно сложно. В то

же время, если сведения о детях выделить в отдельную таблицу, в которой каждому ребенку будет соответствовать отдельная запись, реализация подобных запросов не составит особого труда;

- все «групповые» операции (суммирование, подсчет, определение среднего значения и т.п.) в реляционных СУБД обычно относятся к элементам одного столбца, а не строки; это следует учитывать при проектировании структуры БД;
- для полей типа Memo число допустимых операций по их обработке сильно ограничено по сравнению с полями других типов.

Число подобных примеров можно продолжить.

При проектировании БД необходимо знать особенности языков манипулирования данными в целевой СУБД, а также особенности предполагаемой обработки данных и учитывать это при проектировании структуры БД.

8. *Объем требуемой памяти.* В связи со значительным ростом технических характеристик накопителей и снижением стоимости хранения единицы информации значимость данного фактора постоянно снижается. Исходными данными для определения требуемого объема памяти являются: число объектов отображаемой предметной области, особенности выбранной логической и физической структуры БД, особенности носителя данных. Некоторые CASE-средства включают в себя блоки оценки объемов памяти.

9. *Скорость (время) обработки информации* (время реакции на запрос). Значение данного критерия трудно достаточно точно оценить на стадии проектирования, поскольку на величину этого показателя влияет значительное число взаимосвязанных и взаимозависимых факторов. Если для определения требуемого объема памяти обычно используются аналитические методы, то для определения времени обработки это проблематично. Чаще всего скоростные характеристики определяются путем проведения специальным образом подобранных тестов. Однако факторы, влияющие на скорость обработки, известны, и их следует иметь в виду при проектировании структуры БД.

Рассматриваемый критерий особенно важен для систем, работающих в реальном масштабе времени и в интерактивном режиме.

Перечень приведенных критериев не является исчерпывающим. Кроме того, каждый из перечисленных критериев может быть разбит на множество детализирующих его показателей.

Рассмотрим некоторые примеры, иллюстрирующие оценку тех или иных проектных решений по перечисленным выше критериям. Следует обратить внимание на то, что оценка проекта по какому-либо критерию будет зависеть как от характеристики отображаемой предметной области, так и от особенностей используемой СУБД. Например, некоторые СУБД не позволяют корректировать ключевое поле. Если этого не учитывать и выбрать при проектировании в качестве ключа поле, которое может изменить свое значение, то в случае возникновения в предметной области такой ситуации ее отражение в БД потребует значительных затрат, а именно потребуется перепроектирование структуры БД и довольно трудоемкая процедура перезагрузки данных из старой БД в новую. Если СУБД позволяет корректировать значение

ключевого поля, то таких «перестроек» не потребуется (т.е. показатель оценки одного и того же проектного решения по критерию «устойчивость модели» для разных СУБД будет выглядеть по-разному). Из сказанного не следует делать вывод, что СУБД, позволяющие корректировать ключ, лучше, чем те, которые не позволяют этого: и модель БД получается устойчивее, и проектировать легче (меньше факторов надо учитывать при проектировании). Если выбрать в качестве ключа поле, значение которого может изменяться, то могут возникнуть проблемы при поддержании целостности БД.

При проектировании БД необходимо оценить, с одной стороны, предметную область относительно ее изменчивости, а с другой - проект БД с точки зрения затрат на отображение возможных изменений в предметной области в информационной системе.

Рассмотрим еще один пример, иллюстрирующий оценку проектного решения по некоторым из перечисленных выше критериев.

При ручной организации учета успеваемости в вузах обычно ведется «Книга баллов», в которой каждой группе соответствует своя страница, по строкам которой размещен список студентов, по столбцам - названия дисциплин, которые они изучают в данном семестре. На пересечении строки и столбца проставляется соответствующая отметка.

Если подобную систему перенести без перепроектирования в машинную форму, то это повлечет за собой необходимость создания по меньшей мере стольких файлов/таблиц с разной структурой, сколько имеется разных учебных планов. Число полей в каждом файле будет велико. Без знания учебного плана спроектировать структуру такой БД нельзя. Общее число файлов БД будет еще больше, поскольку для каждого семестра придется делать свой файл, так как в противном случае надо указывать где-то семестр, в котором студент сдавал экзамен; особенно это важно, если по одному и тому же предмету сдается несколько экзаменов.

Каждый раз, когда в учебный план вводятся новые дисциплины, придется менять структуру БД. Обрабатывать БД с такой структурой чрезвычайно тяжело. Например, чтобы знать, какие оценки у студента Х по физике, нужно знать, в какой группе учится данный студент, какая (какие) таблица содержит сведения об его успеваемости, сколько оценок по физике он имеет и как называются соответствующие поля, в которых отражены эти оценки. Многие другие запросы также сформулировать будет сложно. Таким образом, это решение представляется неудовлетворительным по всем основным показателям: плохие адаптивные свойства, велика сложность структуры и обработки. Универсальность системы равна нулю.

Если предложить другой вариант, а именно создать таблицу «Успеваемость» с полями «Код_студента», «Предмет», «Семестр», «Оценка», то все указанные недостатки, отмеченные для первого проектного решения, будут устранены.

Недостатком же второго варианта по сравнению с первым будет большая степень дублирования данных: в первом варианте «ФИО/ Код_студента» фиксируется только один раз как значение поля, а названия предметов вооб-

ще фиксируются только в описании структуры, а во втором варианте они будут повторены как значения соответствующих полей при фиксации каждого факта сдачи экзамена. Но, несмотря на указные недостатки, предпочтение все равно должно быть отдано этому варианту.

Значимость критериев для каждого конкретного проекта будет зависеть от многих объективных и субъективных факторов. Прежде всего она зависит от самого критерия. Среди них есть такие, значимость которых не только никогда не понижается, а наоборот, постоянно растет (например, адекватное отображение действительности, удовлетворение разнообразных потребностей пользователей и т.п.). Актуальность других критериев становится менее существенной с ростом возможностей техники и программного обеспечения (например, занимаемый базой данных объем памяти).

На значимость критерия оценки влияют особенности ИС, для которой создается проект (например, для систем, работающих в реальном масштабе времени, очень важна скорость реакции системы на запросы, для банковских систем - защита информации и надежность системы).

3.2. Особенности даталогических моделей

3.2.1. Внутризаписная структура

В базах данных со структурированными моделями следует различать *внутризаписную* и *межзаписную* структуры. Внутризаписная структура может быть либо *линейной,* либо *иерархической.* При линейной структуре запись состоит из простых элементов, часто называемых полями, которые следуют в записи одно за другим, т.е. структура записи является нормализованной.

В иерархической внутризаписной структуре в состав записи могут входить не только простые, но и составные компоненты. Это могут быть векторы (когда повторяются однотипные элементы), повторяющиеся группы (когда в записи может присутствовать несколько экземпляров составных единиц информации, включающих в себя несколько разнотипных элементов), а также неповторяющиеся составные единицы информации внутри записи. Например, если имеется запись «Личность», то в ее состав могут входить простые элементы, такие, как «Табельный_номер», «Фамилия» и т.д., вектор «Иностранный_язык» (предполагается, что человек может владеть несколькими иностранными языками), повторяющаяся группа «Послужной_список», включающая элементы: «Дата_назначения», «Дата_увольнения», «Место_работы», «Должность», а также неповторяющаяся группа «Адрес», состоящая из элементов «Город», «Улица», «Дом», «Квартира».

Иерархическая структура записи может быть как *одноуровневой,* так и *многоуровневой.* Принципиально возможны довольно сложные структуры, например, когда в состав повторяющейся группы в качестве составляющего компонента входит другая повторяющаяся группа. Однако по разным причинам (в частности, из-за сложности реализации) в конкретных СУБД имеются

различные ограничения (например, повторяющаяся группа может существовать только на первом уровне иерархии, ограничивается число уровней иерархии и т.п.).

Записи могут быть с *постоянным* и *переменным* составом. Последнее обычно понимается так: если значение какого-либо компонента записи отсутствует для конкретного объекта, то и сам этот компонент в данной записи отсутствует. Например, если один из сотрудников окончил вуз, имеет ученую степень и ученое звание, то название вуза, год его окончания, ученая степень, ученое звание и даты их присвоения хранятся в записи, соответствующей данному сотруднику. Если у другого сотрудника все эти признаки отсутствуют, то в соответствующей ему записи эти поля также отсутствуют. Если записи имеют постоянный состав, то все поля, описанные в структуре записи, всегда присутствуют в каждом экземпляре записи, но некоторые из этих полей могут быть пустыми.

Другой характеристикой записи является *тип* ее длины. По этому признаку различают записи с *фиксированной* (постоянной), *переменной* и *неопределенной* длиной. Запись может иметь переменную длину в результате того, что переменную длину имеют ее поля, или возможно отсутствие каких-либо полей, или допускается разное число экземпляров для повторяющихся компонентов.

Поле является наименьшим семантически значимым элементом записи. Основными характеристиками поля являются его тип и длина. Существующие СУБД различаются по набору поддерживаемых типов полей, но наблюдается тенденция к расширению этого набора. Большинство современных СУБД кроме привычных полей символьного и числового типа допускают использование полей типа даты, логических полей, а также полей денежного типа. Некоторые системы позволяют вводить определяемые пользователем типы полей. В последнее время все большее распространение получают мультимедийные формы представления данных.

3.2.2. Межзаписная структура

Традиционное деление СУБД по типу модели данных на реляционные, иерархические и сетевые основывается на характере связей между записями. При всей разнице в терминологии можно считать, что основными компонентами любой из этих моделей являются файлы, которые состоят из записей.

Иерархические модели. В классических иерархических моделях имеется один файл, который является входом в структуру (корень дерева). Остальные файлы связаны друг с другом таким образом, что каждый из них, за исключением корневой вершины, имеет ровно одну исходную вершину («родитель») и любое число подчиненных вершин («детей»). Между записью файла-родителя и записями порожденного файла имеется отношение 1:M (как частный случай может быть и отношение 1:1). Имеются и другие разновидности иерархических моделей, но они менее распространены.

Сетевые модели. Если на сетевые модели не накладывается никаких ограничений, в принципе любой файл может быть точкой входа в БД, каждый из файлов может быть связан с произвольным числом других файлов и между записями связанных файлов могут быть любые отношения (1:1, 1:M, M:M). Однако в реальных СУБД на модель накладываются различные ограничения. Так, имеются сетевые СУБД с разнотипными файлами. В них все файлы разделены на два типа: основные и зависимые. В таких СУБД входом в базу данных Могут служить только основные файлы, а связываться между собой могут только разнотипные файлы.

Во многих сетевых СУБД не поддерживается непосредственно отношение M:M. В таких моделях каждая связь между парой файлов определяется отдельно, и для каждой из них один файл в этой паре объявляется владельцем, а другой - членом. Отношение между записью-владельцем и записями-членами - 1:M.

Связи между файлами в иерархических и сетевых моделях определяются при описании структуры базы данных и физически чаще всего передаются с помощью различных указателей.

Реляционные модели. В таких моделях используется своеобразная терминология, но это не меняет сущности модели. Часто даже в рамках одной модели в разных СУБД используется разная терминология. Так, на логическом уровне элемент чаще всего называют атрибутом; кроме того, для него используются термины «колонка», «столбец», «поле». Совокупность атрибутов образует строку (синонимичные термины - «ряд», «запись», «кортеж»). Совокупность строк образует отношение («таблицу», «файл базы данных»). Понятие базы данных как множества отношений поддерживается далеко не всеми реляционными СУБД (т.е. при создании БД описываются отдельные отношения (файлы, таблицы), а для всей базы данных как самостоятельной информационной единицы никакого описания не предусмотрено). Однако следует отметить, что в последнее время новые версии даже тех настольных СУБД, которые раньше не поддерживали понятие БД, сейчас его включают.

Связи между файлами в реляционной модели в явном виде могут не описываться. Они устанавливаются динамически в момент обработки данных по равенству значений соответствующих полей.

В сетевых и иерархических моделях структура записи в принципе может быть любой, хотя многие СУБД накладывают различные ограничения на структуру записи. В реляционных моделях структура записи должна быть линейной.

Каждое отношение по определению имеет *ключ,* т.е. атрибут (простой ключ) или совокупность атрибутов (составной ключ), однозначно идентифицирующих кортеж. В некоторых случаях в отношении могут быть несколько *возможных (вероятных, альтернативных) ключей.*

К сожалению, не все реляционные СУБД поддерживают концепцию ключа, поскольку в этом случае многие проблемы (в частности, обеспечение проверки на уникальность ключа и соблюдение некоторых других ограничений целостности) возлагаются на пользователя (проектировщика ИС). Вне

зависимости от того, требует ли СУБД явного указания ключей при описании отношений или нет, проектировщик базы данных должен понимать, что является ключом каждого отношения.

В случае если СУБД поддерживает концепцию ключа, при наличии нескольких вероятных ключей один из них выбирается и описывается как *первичный,* остальные — как *альтернативные* (обычно для этих целей используется либо уникальный индекс, либо описатель UNIQUE при определении таблицы).

Атрибут (или группа атрибутов), который в рассматриваемом отношении не является ключом, а в другом отношении - является, называется *внешним ключом.*

Если какая-либо таблица содержит внешний ключ, то она логически связана с таблицей, содержащей соответствующий первичный ключ; эта связь имеет характер «один ко многим» (таблица, содержащая внешний ключ, находится на стороне «много» в этой связи). В частном случае это может быть связь 1:1. Связь M:M в реляционной базе данных непосредственно не поддерживается.

По сути, понятия «родитель» - «ребенок» в иерархических моделях, «файл-владелец» - «файл-член» в сетевых моделях и связь «ключ» - «внешний ключ» в реляционных моделях передают одно и то же явление — наличие связи 1 :M между записями соответствующих файлов.

В реляционных СУБД часто используется понятие *взгляд (view).* Оно трактуется как виртуальная таблица, часто полученная в результате логического соединения нескольких связанных по значениям общих столбцов таблиц и, возможно, включающая некоторое подмножество из всей совокупности строк, отобранное по заданному условию. Это понятие расширяет традиционное для банков данных понятие «подсхема».

Понимание различий и общности моделей разных классов позволяет использовать общий подход при проектировании структуры баз данных, осуществлять преобразование одних моделей в другие, использовать средства, в частности языковые, предназначенные для работы с одним классом моделей, при работе с другим.

Системы, построенные на основе инвертированных файлов. Это еще одна разновидность моделей данных. Относительно логической структуры БД такие модели следует отнести к сетевым. Более того, это единственный вид моделей, которые в явном виде поддерживают отношение M:M между файлами БД. Особенности этих моделей заключены в способе отражения связей между информационными единицами. В системах, построенных на основе инвертированных файлов, собственно хранимые данные и информация о связях между информационными единицами логически и физически отделены друг от друга. Основные данные в этих системах хранятся в файлах, записи которых могут иметь многоуровневую иерархическую структуру. Вся управляющая информация сосредоточена в так называемом ассоциаторе. Логическая связь между файлами устанавливается посредством компонента ассоциатора, называемого сетью связи. Отделение ассоциативной информа-

ции от собственно хранимых данных позволяет изменять связи, не изменяя при этом самих файлов.

3.3. Проектирование логической структуры реляционной базы данных

3.3.1. Вводные положения

Для реляционной базы данных проектирование логической структуры заключается в том, чтобы разбить всю информацию по файлам* (или в терминах реляционной модели - по отношениям, таблицам), а также определить состав полей (в терминах реляционной теории -атрибутов) для каждого файла. Определение ключа каждого из отношений также является задачей логического проектирования реляционной БД.

Сейчас многие реляционные СУБД позволяют декларативно задавать связи между таблицами при описании БД, а также определять необходимость контроля и способы обеспечения целостности по связям для БД. Решение этих вопросов также следует отнести к даталогическому проектированию. Другие ограничения целостности (кроме ограничения на уникальность и ограничения по связи) в данном разделе рассматриваться не будут.

Часто при описании логической структуры реляционной БД сразу же указывается, по каким полям надо индексировать соответствующий файл, а для ключевых полей автоматически предусматривается индексация. Индексация занимает промежуточное положение между логической и физической структурой данных: с одной стороны, она определяет способ упорядочения данных и доступ к ним, а с другой - это способ «логического упорядочения», при котором создаются вспомогательные индексные файлы, что меняет общую структуру БД. Вопросы индексирования будут частично рассмотрены в данном разделе.

Существуют разные методы проектирования логической структуры реляционных баз данных. Среди них есть и строгие математические методы, обычно базирующиеся на теории нормализации. Они имеют очень большое значение в качестве теоретической основы проектирования БД, но в связи с вычислительной сложностью алгоритмов практически не используются в реальном проектировании систем.

Рассмотрим метод проектирования, основанный на анализе ER-модели и переходе от нее к реляционным отношениям. В основу этого метода положен эмпирический подход. Предлагаемый метод является достаточно простым и наглядным и в то же время дает хорошие результаты. Базы данных, полученные в результате применения излагаемой ниже методики проектирования, находятся в 4-й нормальной форме. Следует отметить, что большинство имеющихся в настоящее время CASE-средств также используют аналогич-

* В некоторых СУБД каждой таблице ставится в соответствие файл базы данных, в других - вся база данных хранится в одном файле. Здесь термины «файл», «таблица» и «отношение» используются как синонимы и не отражают способ физического хранения данных

ный подход, но сам алгоритм проектирования обычно нигде не публикуется и может быть воспроизведен лишь экспериментально, путем анализа проектных решений, полученных при преобразовании тех или иных ER- конструкций в схему целевой БД.

3.3.2. Алгоритм перехода от базовой ER-модели к схеме реляционной базы данных

Каждый элемент ER-модели находит свое отражение в схеме базы данных. Для некоторых ситуаций в ER-модели возможно использование нескольких альтернативных решений при их отображении в модель базы данных. Выбор наиболее подходящего решения будет зависеть от разных факторов, часть из которых отображена в ER-модели, а другая часть - нет, т.е. для выбора проектного решения кроме информации непосредственно из ER-модели необходимо дополнительно использовать информацию и из других компонентов концептуальной модели предметной области.

Рассмотрим рекомендации по переходу от базовой ER-модели к схеме реляционной базы данных для каждого типа элементов ER-модели.

Отображение простых объектов

Определение состава полей основной таблицы. Для каждого простого объекта и его единичных свойств строится отношение, атрибутами которого являются идентификаторы объекта и реквизиты, соответствующие каждому из единичных свойств.

Для объекта O_1 (*см.* рис. 2.62, а) отношение может выглядеть следующим образом:

$$Rl\ (\underline{ИО_1},\ C_1,\ C_2,\ C_5,\ C_6). \qquad (3.1)$$

Определение ключа таблицы. Любой из уникальных идентификаторов объекта является вероятным ключом полученного отношения. Если объект имеет несколько уникальных идентификаторов, один из них выбирается в качестве первичного ключа, как правило, самый короткий из вероятных ключей. Но это не всегда должно быть обязательно так. На решение вопроса о выборе первичного ключа (кроме длины ключа) влияет ряд факторов.

1. *Стабильность* - может ли значение ключа изменяться. Несмотря на то, что многие современные СУБД не только не запрещают изменять значение первичного ключа, но и имеют специальные механизмы, позволяющие автоматически осуществлять изменения связанных записей (каскадное изменение), не следует злоупотреблять этой возможностью. Желательно выбирать в качестве первичного ключа атрибуты, которые не изменяются.

2. Мнемоничность - легкость запоминания. Поскольку в качестве ключа обычно используются короткие обозначения, то при прочих равных условиях следует отдавать предпочтение тем из вероятных ключей, которые легче запомнить.

Среди названных выше критериев наиболее важным является стабильность. Например, если у объекта КАФЕДРА есть три идентификатора: «Наименование_кафедры_полное», «Наименование_кафедры_краткое» и «Код_кафедры», то даже если «Краткое_наименование» короче других полей, скорее всего в качестве первичного ключа будет выбираться «Код», потому что наименования кафедр подвержены достаточно частым изменениям. Поэтому в алгоритме при выборе первичного ключа в качестве «решения по умолчанию» принимается более короткий идентификатор, но от пользователя требуется либо подтвердить это решение, либо выбрать иной первичный ключ.

Если объект является *зависимой по идентификации сущностью,* то ключ соответствующего ему отношения будет составным, включающим идентификатор этого объекта и идентификатор «вышестоящего» объекта, т.е. идентификатор основного объекта мигрирует в таблицу, соответствующую зависимому объекту (рис. 3.1). Если идентификаторов у главного объекта несколько, то выбирается один из них, а именно тот, который выбран в качестве первичного ключа основной сущности. Полученный таким образом составной идентификатор зависимой по идентификации сущности будет использоваться во всех случаях, когда нужно отображать связь этого зависимого объекта с другими.

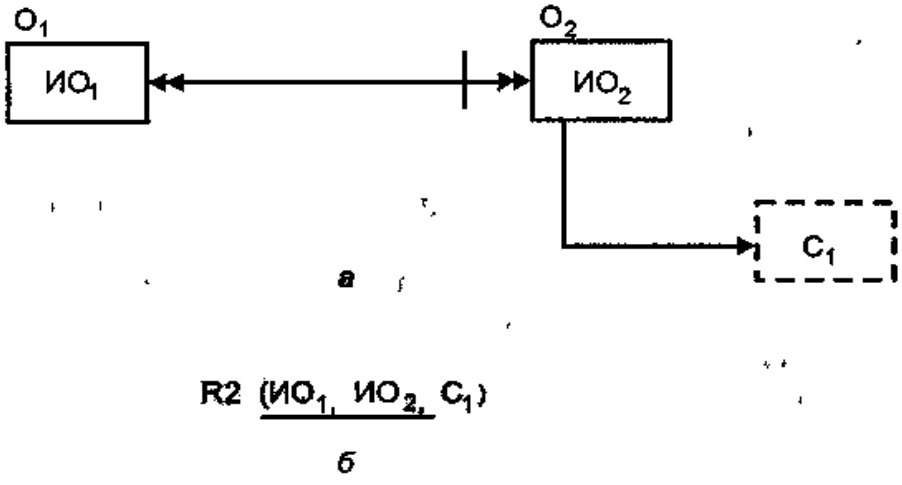


Рис. 3.1. Отображение зависимой по идентификации сущности:
а - фрагмент ER-модели; *б* - реляционная таблица

Обычно зависимый по идентификации объект и тот объект, от которого он зависит, связаны друг с другом отношением 1:M, и может сложиться впечатление, что перенесение ключа просто соответствует отображению этой связи. Однако это не совсем так. Если сущность не зависима по идентификации, то связь 1:M можно отображать в структуре БД как путем переноса ключа связанного объекта в таблицу, соответствующую подчиненному объекту (т. е. объекту, стоящему со стороны M), так и другими способами, а ключом таблицы, соответствующей подчиненному объекту, будет являться только идентификатор самого этого объекта. Для зависимого по идентификации объекта связь 1:M дополнительно в схеме БД отображать не надо.

Некоторые СУБД (например, Access, Paradox и др.) позволяют автоматически генерировать поле типа «счетчик» в качестве ключа таблицы. Этот искусственный код вполне можно создавать для простых объектов, если в предметной области не предполагается использование другой системы кодирования объектов (например, для предприятий или иных объектов хозяйственной деятельности в БД все равно в большинстве случаев приходится хранить коды ОКПО, ОКОНХ, ИНН; в этом случае создавать дополнительный код может не иметь смысла). Созданные коды будут в дальнейшем использоваться для связи данного объекта с другими объектами в БД.

Если такой код применять при создании таблицы, соответствующей агрегированному объекту, то вряд ли он будет где-то использоваться в дальнейшем, хотя это утверждение нельзя рассматривать категорично. Например, для агрегированного объекта СДАЧА_ЭКЗАМЕНА каждая попытка может иметь дополнительную информацию (какие вопросы были заданы и т.п.). В этом случае для отображения этой информации может потребоваться отдельная таблица и код «сдачи экзамена» может оказаться полезным.

Другими словами, при создании таблицы в каждом конкретном случае необходимо решать вопрос, что выбрать в качестве ключа: естественный ключ, искусственный ключ, в том числе и созданный системой автоматически, а может быть, если СУБД это позволяет, отказаться от создания ключа вообще.

Отображение множественных свойств объекта. Если у объекта имеются множественные свойства, то каждому из них ставится в соответствие отдельное отношение, полями которого будут идентификатор объекта (при наличии у объекта нескольких идентификаторов - тот из них, который выбран в качестве первичного ключа) и поле, отображающее множественное свойство. Ключ этого отношения будет составным, включающим оба эти атрибута. Так, для множественных свойств $С_3$, $С_4$ (*см.* рис. 2,62, *а)* будут созданы отношения

$$R2\ (\underline{ИО_1}, \underline{С_3});$$

$$R3\ (\underline{ИО_1}, \underline{С_4}).$$

Приведенное выше решение является универсальным. В отдельных случаях могут быть приняты и другие варианты. Так, если число экземпляров множественного свойства у каждого объекта невелико и в процессе обработки не возникает необходимости выделять каждое из этих значений, то можно все значения, относящиеся к одному объекту, хранить в одном поле. В этом случае отдельную таблицу для хранения множественного свойства создавать не нужно.

Отображение условных свойств объекта. Если объект обладает условными свойствами, то при отображении их в реляционную модель возможны варианты.

1. Если многие объекты обладают рассматриваемым свойством, то его можно хранить в базе данных так же, как и обычное свойство, т.е. в той же

таблице, в которой бы атрибут хранился, если бы свойство было определенным для всех экземпляров рассматриваемой сущности.

2. Если только незначительное число объектов обладает указанным свойством, то для многих записей в файле базы данных при использовании предыдущего решения значение соответствующего поля будет пустым. Для устранения этого недостатка можно выделить отдельное отношение, которое будет включать идентификатор объекта и атрибут, соответствующий рассматриваемому свойству. Это отношение будет содержать столько строк, сколько объектов имеет рассматриваемое свойство. Однако это решение, в свою очередь, имеет недостатки (в частности, усложнение структуры БД и трудности ее обработки) и применяется сравнительно редко.

В отношении (3.1) использован вариант 1. Если бы было выбрано второе из обсуждаемых решений, то из отношения R1 атрибут C_5 следовало бы исключить и создать дополнительно новое отношение:

$$R4\ (\underline{ИО}_1, C_5). \qquad (3.2)$$

Отображение составных свойств объекта. Если объект имеет составное свойство, то возможны два варианта его отображения в БД:

1. Всему составному свойству ставится в соответствие одно поле (3.1).

2. Каждому из составляющих элементов составного свойства ставится в соответствие отдельное поле:

$$Rl\ (\underline{ИО}_1, C_1, C_2, C_5, C_7, C_8). \qquad (3.3)$$

Выбор варианта будет зависеть в основном от характера преимущественной обработки этой информации. Поскольку в большинстве СУБД гораздо проще при реализации запросов объединить поля, чем выделить из единого поля нужную часть, то в случае, если предполагается использование отдельных компонентов составного свойства, лучше выбрать вариант 2, в противном случае - вариант 1.

Отображение связи между объектами

Связи между объектами также должны отражаться в структуре БД. Универсальный способ отображения связи между объектами - введение вспомогательного связующего файла, содержащего идентификаторы связанных объектов. Ключ этого отношения будет составным. Такое решение является практически единственно приемлемым при наличии связи M:M между объектами. Дополнительный довод в пользу такого решения - наличие необязательного класса членства объекта в связи. Во многих случаях можно использовать другие, более эффективные способы отображения связей в структуре БД. Выбор проектного решения прежде всего будет зависеть от типа связи между объектами.

Отображение связи типа M:M. Если между объектами предметной области имеется связь M:M, то для хранения такой информации потребуются три отношения: по одному для каждой сущности и одно дополнительное - для отображения связи между ними. Последнее отношение будет содержать идентификаторы связанных объектов (рис. 3.2). Ключ этого отношения будет составным.

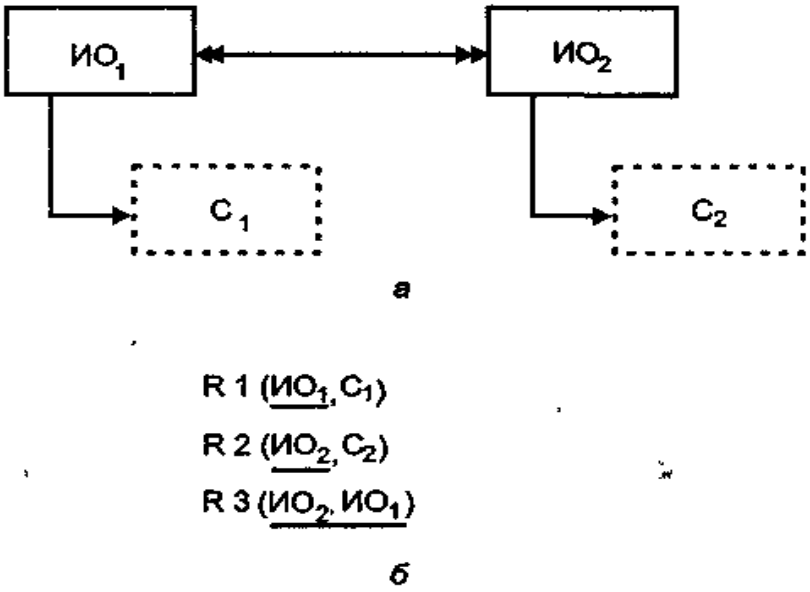
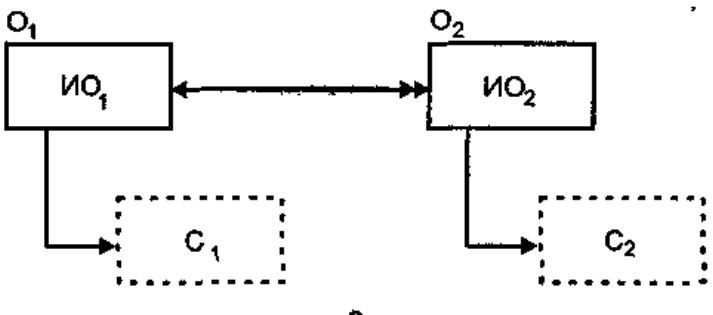


Рис. 3.2. Отображение связи M:M: *а* - фрагмент ER-модели;
б - реляционные таблицы

Отображение связи типа 1:M. Если между объектами предметной области имеется связь 1:M, то можно, так же как и в случае связи M:M, использовать отдельную связующую таблицу (рис. 3.3, *б* - вариант 2). В отличие от связи M:M ключом связующей таблицы будет только идентификатор объекта, к которому направлен единичный конец связи.

Однако если между объектами предметной области имеется связь 1:M и класс принадлежности n-связной сущности является обязательным, то можно использовать только два отношения (по одному для каждой сущности) и не использовать дополнительную связующую таблицу. В отношение, соответствующее 1-связной сущности (т.е. сущности, к которой идет единичная связь), нужно дополнительно добавить идентификатор связанного с ней объекта (рис. 3.3, б - вариант 1).

Если класс принадлежности n-связной сущности является необязательным, то появляется дополнительный довод в пользу решения о создании для отображения связи третьего отношения, которое будет содержать ключи каждой из связанных сущностей (рис. 3.3, б - вариант 2).



а

Вариант 1

R 1 (ИО$_1$, C$_1$)
R 2 (ИО$_2$, C$_2$, ИО$_1$)

Вариант 2

R 1 (ИО$_1$, C$_1$)
R 2 (ИО$_2$, C$_2$)
R 3 (ИО$_1$, ИО$_2$)

б

Рис. 3.3. Отображение связи 1:М: *а* - фрагмент ER-модели;
б - реляционные таблицы

Отображение связи типа 1:1. Наличие между объектами связи типа 1:1 является довольно-таки редкой ситуацией в реальной жизни. В принципе, если связь между объектами 1:1 и класс принадлежности обеих сущностей является обязательным, для отображения обоих объектов и связи между ними можно использовать одну таблицу (рис. 3.4, *б* - вариант 3). Такое решение потребует меньше всего памяти для своей реализации. Например, если имеются объекты СОТРУДНИК и ПАСПОРТ, то такое решение будет вполне приемлемым. Однако таким решением не следует злоупотреблять. Может случиться, что для каждого объекта, находящегося в связи 1:1, в дальнейшем понадобится отразить какие-то свои связи или в запросах потребуется информация отдельно по каждому из объектов. Тогда выбранное решение может усложнить или замедлить работу с БД.

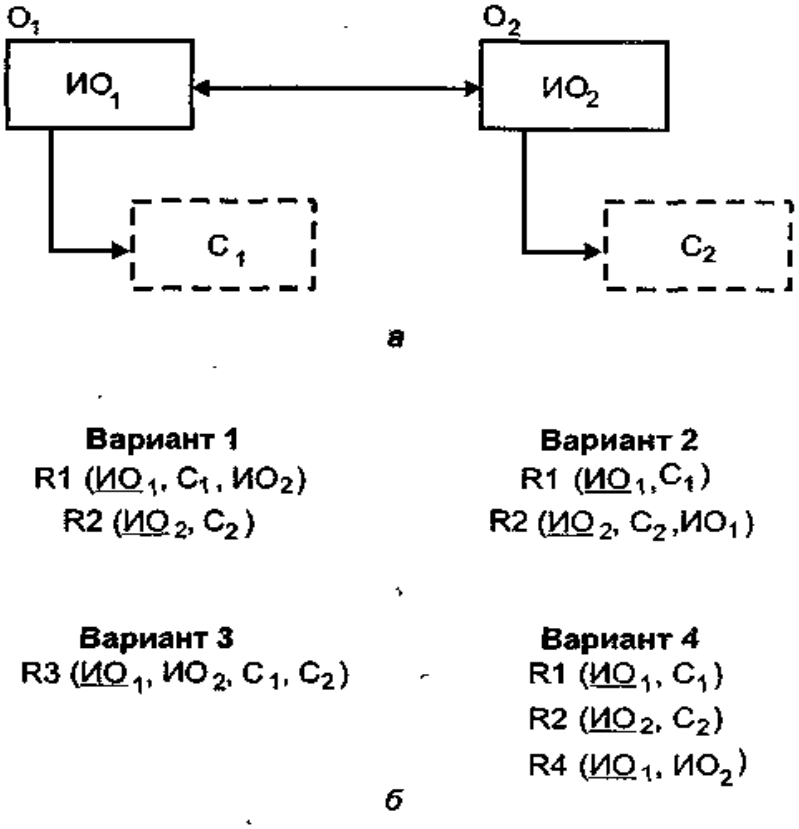


Рис. 3.4. Отображение связи 1:1: *а* - фрагмент ER-модели;
б - реляционные таблицы

Если для каждого такого объекта создаются отдельные отношения, то информацию о связях между ними можно отразить, включив в одно из отношений идентификатор связанного объекта из другого отношения. Причем если класс принадлежности обеих сущностей является обязательным, то (если руководствоваться только типом связи) это можно сделать в любом из отношений (рис. 3.4, б - варианты 1 и 2).

Если класс членства одного из объектов является необязательным, то идентификатор сущности, для которой класс принадлежности является необязательным, добавляется в отношение, соответствующее тому объекту, для которого класс принадлежности - обязательный.

Если степень связи между объектами равна 1:1 и класс принадлежности каждого из них является необязательным, то, чтобы избежать наличия пустых полей, следует использовать три отношения: по одному для каждой сущности и одно - для отображения связи между ними (рис. 3.4, *б* - вариант 4). В приведенном решении в качестве ключа связующей таблицы обозначен $ИО_1$. С таким же успехом мог быть выбран $ИО_2$.

Отображение альтернативной связи. Альтернативная связь обычно используется при изображении агрегированного объекта и означает, что в действии участвует либо один объект, либо другой, но не оба вместе. Альтернативная связь трудна для ее «автоматического» преобразования в дата-

логическую модель. Может быть, поэтому она отсутствует в большинстве CASE-средств.

Естественным кажется путь, когда в таблице базы данных, соответствующей объекту, к которому идут альтернативные связи, всем этим связям будет соответствовать одно поле и в нем будет зафиксирован идентификатор связанного объекта. В экземпляре записи в этом поле будет записано значение идентификатора того объекта, который участвует в отображаемой связи в каждой конкретной ситуации. Но это решение имеет множество недостатков, связанных с последующей обработкой таким образом спроектированных таблиц, и его в большинстве случаев не рекомендуется использовать.

Другой вариант решения: для отражения связи с каждым из альтернативных классов объектов использовать отдельную таблицу.

Часто объекты, объединенные альтернативной связью, по сути, являются подклассами обобщенного класса. Иногда имеет смысл по-иному изобразить ER-модель, чтобы увидеть другие варианты проектных решений.

Отображение сложных объектов

Выше были рассмотрены варианты проектных решений, связанные с простыми объектами. Но в ER-модели отражаются и сложные объекты.

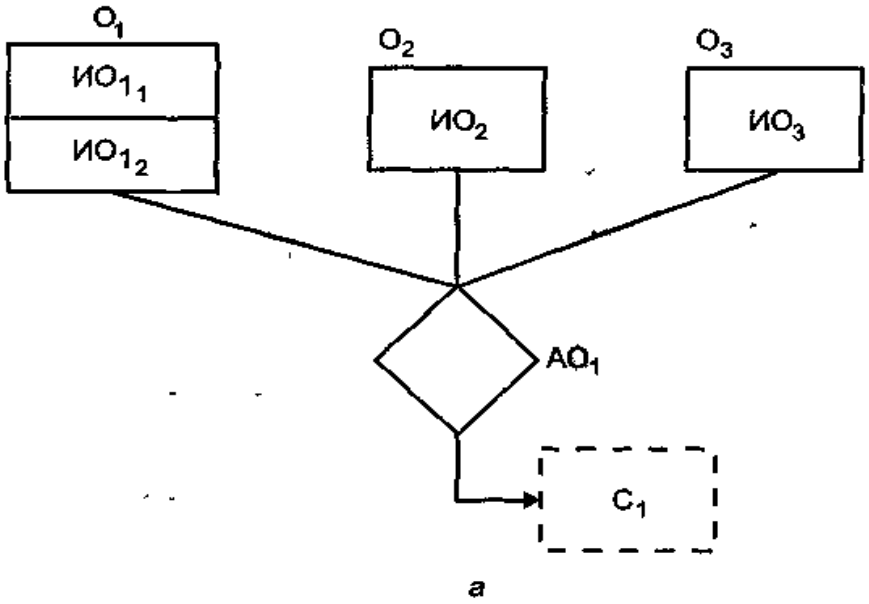
Отображение агрегированных объектов. Каждому агрегированному объекту, имеющему место в предметной области, в реляционной модели будет соответствовать отдельное отношение. Атрибутами этого отношения будут являться идентификаторы всех объектов, задействованных в данном агрегированном объекте, а также реквизиты, соответствующие свойствам этого агрегированного объекта.

Для отношений, соответствующих агрегированным объектам, ключ будет составной. В большинстве случаев им будет являться конкатенация (соединение) идентификаторов объектов, участвующих в этом агрегированном объекте (рис. 3.5).

Объединить информацию о нескольких агрегированных объектах в одно реляционное отношение можно только в случае, если те объекты, с которыми связан каждый из них, полностью совпадают (рис. 3.6). Это является необходимым, но не достаточным условием для такого объединения. В каждом конкретном случае возможность и необходимость такого объединения нужно определять особо.

Отображение обобщенных объектов. При этом могут быть приняты разные решения.

Во-первых, всему обобщенному объекту может быть поставлена в соответствие одна таблица базы данных (рис.3.7, *б* - вариант 1). В этом случае атрибутами таблицы будут идентификаторы обобщенного объекта и все единичные свойства, присущие объектам хотя бы одной категории, включая свойство, по которому проводится разбиение на подклассы. Ключом таблицы будет один из идентификаторов этого объекта.

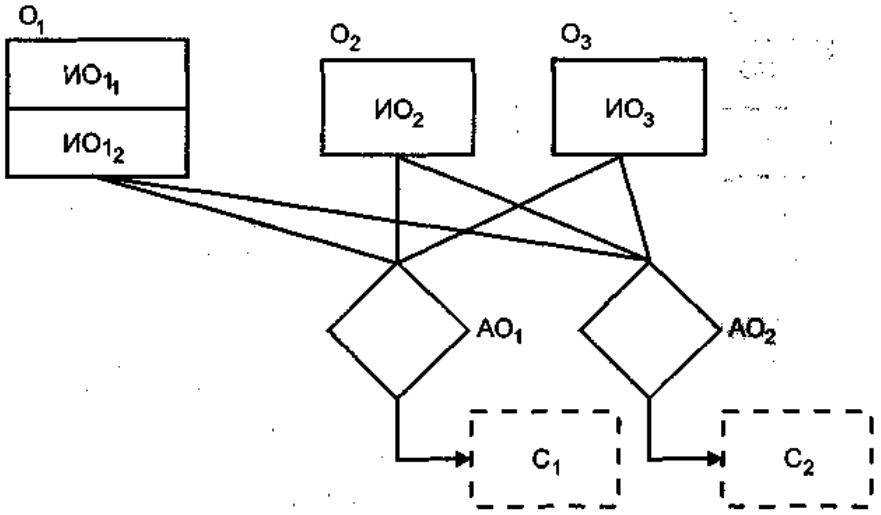


а

RAO_1 ($\underline{ИО_{1_1}}$, $\underline{ИО_2}$, $\underline{ИО_3}$, C_1)

б

Рис. 3.5. Отображение агрегированного объекта:
а - фрагмент ER-модели; *б* - реляционная таблица



а

Вариант 1

RAO_1 ($\underline{ИО_{1_1}}$, $\underline{ИО_2}$, $\underline{ИО_3}$, C_1)

RAO_2 ($\underline{ИО_{1_1}}$, $\underline{ИО_2}$, $\underline{ИО_3}$, C_2)

Вариант 2

RAO_1 ($\underline{ИО_{1_1}}$, $\underline{ИО_2}$, $\underline{ИО_3}$, C_1, C_2)

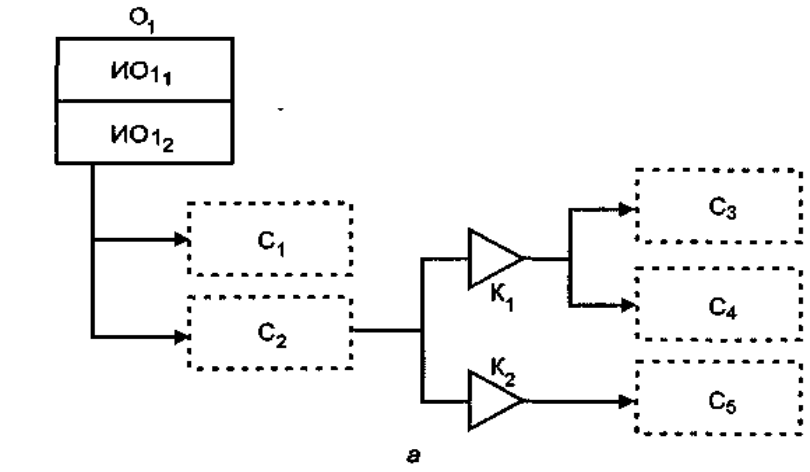
б

Рис. 3.6. Отображение нескольких агрегированных объектов, имеющих одинаковые связи: *а* - фрагмент ER-модели; *б* - реляционные таблицы

Другим «крайним» вариантом является решение, при котором каждой категории объектов нижнего уровня ставится в соответствие отдельное отношение (рис.3.7, *б* - вариант 2). В этом случае каждое отношение будет включать в себя идентификатор объекта (если идентификаторов несколько, то в каждое из отношений будут включены все они; это не приведет к дублированию информации на уровне значений), свойства, присущие родовым объектам, а также свойства, присущие данному подвиду объектов. Свойство, по которому проводится разбиение класса на подклассы, в этом случае в качестве поля не включается ни в одно из отношений.

Кроме этих двух крайних решений возможны и комбинированные варианты. Например, можно выделить общую таблицу для отображения родовых свойств объектов (включающую еще и все идентификаторы объекта) и отдельные таблицы для отображения видовых свойств (такой алгоритм используется в системе Design/IDEF). Кроме свойств, присущих видовому объекту, в каждом из этих отношений будет повторен ключевой атрибут основного отношения (рис. 3.7, *б* - вариант 3).

Другим вариантом проектного решения для отображения обобщенного объекта является использование так называемого кодированного формата файла, при котором, так же как и в варианте 1, используется одна таблица, но для всех видовых свойств каждого подкласса выделяется одно поле. Его содержимое распознается по значению свойства, по которому проводится разбиение класса на подклассы.



а

Вариант 1

R1(ИО11, ИО12, C1, C2, C3, C4, C5)

Вариант 2

R1 (ИО11, ИО12, C1, C3, C4)
R2 (ИО11, ИО12, C1, C5)

Вариант 3

R1(ИО11, ИО12, C1, C2)
R2 (ИО11, C3, C4)
R3 (ИО11, C5)

б

Рис. 3.7. Отображение обобщенного объекта:
а - фрагмент ER-модели; *б* - реляционные таблицы

Перечень вариантов можно продолжить. Выбор конкретного решения будет зависеть от многих факторов, в том числе от того, насколько часто информация о разных категориях объектов обрабатывается совместно, как велико различие в видовых свойствах и др.

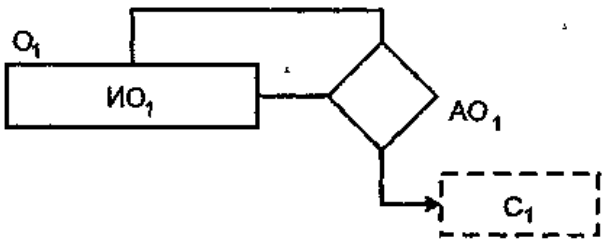
Приведенный выше алгоритм излагался из предположения, что классификация объектов не являлась фасетной. Если в обобщенном объекте наблюдается разбиение на подклассы по разным несоподчиненным признакам, то варианты 1 и 3 останутся верны, а вариант 2 должен быть уточнен.

Кроме того, алгоритм не учитывает, что классы могут быть пересекающимися.

Кроме того, при выборе проектного решения необходимо учитывать, является класс полным или нет. Если класс неполный, то при выборе варианта, когда для каждого подкласса строится отдельная таблица, информация об объектах, не вошедших ни в один подкласс, может просто пропасть.

Отображение составных объектов. В базовой ER-модели, как и в большинстве других нотаций, нет специальных обозначений для отображения связи «целое-часть» или составного объекта (*см.* главу 2).

Наличие такой связи может быть отображено как в инфологической, так и в даталогической модели по-разному. Следует отметить, что само отношение «целое-часть» может качественно различаться для разных ситуаций. Так, если речь идет о составе изделий, то между ИЗДЕЛИЕМ и ДЕТАЛЬЮ имеется связь типа M:M, так как одна и та же деталь может входить в разные изделия и, наоборот, в изделие входят разные детали. Состав изделия обычно является сложным, и отражать его в явном виде в структуре базы данных нежелательно, а часто и просто невозможно. Кроме того, рассматриваемая связь реализована на однородном множестве объектов. В этом случае для отображения связи «целое-часть» можно воспользоваться двумя файлами базы данных. Первый из них будет хранить информацию о самих объектах, а второй - информацию о связи между ними, а также дополнительную информацию, характеризующую эту связь. Для состава изделия это могут быть поля «что входит», «куда входит» и «количество» (рис. 3.8).



а

R (ИО1₁, ИО1₂, С1)

б

Рис. 3.8. Отображение состава изделия:
а - фрагмент ER-модели; *б* - реляционная таблица

Отношение «целое-часть» может отражать, например, структуру какой-то организации. В этом случае ему, скорее всего, будет соответствовать связь типа 1:M, и для его отображения в даталогической модели можно использовать рекомендации, данные выше для соответствующего случая. В рассматриваемой ситуации также можно воспользоваться неявным выделением уровней, но такой прием используется при отображении организационной структуры редко.

Использование дополнительных характеристик концептуальной модели

Характеристики свойств динамические (Д) и статические (С) могут быть использованы при задании ограничений целостности (например, можно задать запрет на обновление для статических полей). Кроме того, эта информация может быть полезна при выборе ключа отношения, а также способов физической организации данных.

Характеристики «число объектов», «рост числа объектов», «летучесть» могут быть использованы для управления размещением данных на носителе, выбора методов организации данных и доступа к ним, определения объемных характеристик БД (что относится к стадии физического проектирования БД). Эти характеристики также могут быть приняты во внимание при решении вопросов о целесообразности разбиения файла на несколько самостоятельных файлов.

«Класс членства» объектов в связи оказывает влияние не только на выбор варианта построения логической структуры, но и на задание ограничений целостности. (Более подробно об этом *см.* в главе 4.)

Информация о степени пересечения классов объектов (граф пересечений) может учитываться при выборе варианта отображения обобщенного объекта, а также при контроле целостности базы данных.

Дополнительные рекомендации по проектированию БД

Реляционная база данных, полученная в результате использования предложенного выше алгоритма проектирования, является нормализованной и автоматически находится в четвертой нормальной форме (4 НФ).

Как известно, в принципе вся предметная область может быть представлена в виде одного универсального отношения. Недостатки такого способа хранения известны, и нормализация отношений служит устранению этих недостатков. Нормализация выполняется путем вертикального разбиения (проекции) исходного отношения.

Однако хранение нормализованных файлов, в свою очередь, может привести к отрицательным последствиям, например к замедлению выполнения некоторых операций. Поэтому иногда сознательно идут на денормализацию отношений.

Нормализованные отношения, в свою очередь, могут быть подвергнуты вертикальному разбиению, например в случае, если информация, хранящаяся в одном отношении, редко обрабатывается совместно или если ограничения СУБД не позволяют хранить все атрибуты в одном отношении. Вопросы вертикального разбиения нормализованных отношений выходят за рамки теории нормализации, но их часто приходится рассматривать в реальной практике проектирования ИС. Также за рамками теории нормализации остаются вопросы горизонтального разбиения отношений.

На решение вопросов о необходимости и степени вертикального и горизонтального разбиения отношений оказывает влияние много факторов, напрямую связанных не только с оценкой структуры данных (объем занимаемой памяти, степень дублирования данных), но и с затратами времени на выполнение операций разного типа, а также с корректностью получаемого при этом результата.

Горизонтальное разбиение в принципе может давать как непересекающиеся, так и пересекающиеся множества. При получении непересекающихся множеств степень дублирования данных не изменится, и объем памяти, занимаемый под данные, останется таким же, как и до расщепления. Однако общий объем памяти, требуемый для хранения БД, возрастет. Это произойдет за счет большего объема служебной информации (каждое отношение должно быть самостоятельно описано), а также за счет неиспользованного свободного места в конце каждого физического блока (сектора). Кроме того, возрастет сложность базы данных (одним из показателей которой является число информационных единиц в структуре БД) и сложность обработки. При пересекающихся множествах, кроме того, возрастет степень дублирования данных и возникнут проблемы с поддержанием целостности данных.

Несмотря на очевидные недостатки горизонтального разбиения, к такому приему довольно часто прибегают в практике проектирования. Чем это бывает вызвано? Во-первых, горизонтальное разбиение может обеспечить сокращение времени обработки данных, в том числе и за счет распараллеливания выполнения операций. Так, выполнение операций селекции и проекции над горизонтальными сечениями эквивалентно выполнению этих операций над всем отношением, и они могут осуществляться параллельно. При этом общее требуемое время выполнения запроса будет меньше, чем t / p, где t - время выполнения запроса над всем отношением; p - число доступных процессоров, так как время еще может сократиться и за счет работы с более короткими файлами.

Однако не все операции можно непосредственно выполнить над горизонтальными сечениями отношений. Рассмотрим следующий пример. Пусть в БД учебного института хранится информация о студентах. Основными потребителями этой информации являются деканаты соответствующих факультетов. Информация чаще всего обрабатывается и анализируется в пределах факультетов. В этом случае целесообразно сделать горизонтальные сечения и хранить данные в разрезе каждого факультета отдельно. Однако если,

например, потребуется получить средний балл успеваемости всех студентов в последнюю сессию, то придется либо объединить соответствующие файлы, либо использовать достаточно сложный алгоритм вычисления требуемого показателя.

Горизонтальное разбиение отношения может проводиться по разным принципам. В качестве условия разбиения может использоваться значение какого-либо атрибута (как, например, в рассмотренном выше примере - значение атрибута ФАКУЛЬТЕТ). Довольно часто используется разбиение по временному признаку, например данные за каждый месяц хранятся в отдельном файле. Могут использоваться и другие принципы разбиения: по достижению определенного объема файла, по активности записей и т.д.

3.4. Создание физической модели в ERWin

3.4.1. Выбор целевой СУБД

Как было отмечено ранее, в ERWin представление структуры базы данных для конкретной целевой СУБД называется физической моделью.

Для отображения физической модели в ERWin следует воспользоваться списком выбора, расположенным в левой части панели инструментов (рис. 3.9).

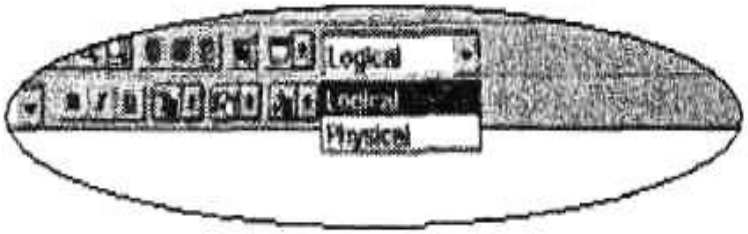


Рис. 3.9. Переключение в режим отображения физической модели

Перед созданием физической модели следует выбрать целевую СУБД. Для этого можно воспользоваться кнопкой  (Select Target Server) инструментального меню. В появившемся окне (рис. 3.10) нужно выбрать желаемую СУБД и ее версию. Эта возможность доступна только на физическом уровне.

Если в явном виде СУБД не указана, то физическая модель строится для СУБД, выбираемой по умолчанию (Oracle). Сменить целевую СУБД можно на любом этапе проектирования. Возможность использовать одну и то же логическую модель для получения разных физических моделей является большим преимуществом CASE-технологий.

Вид окна **Target Server** будет несколько различаться в зависимости от выбранной СУБД.

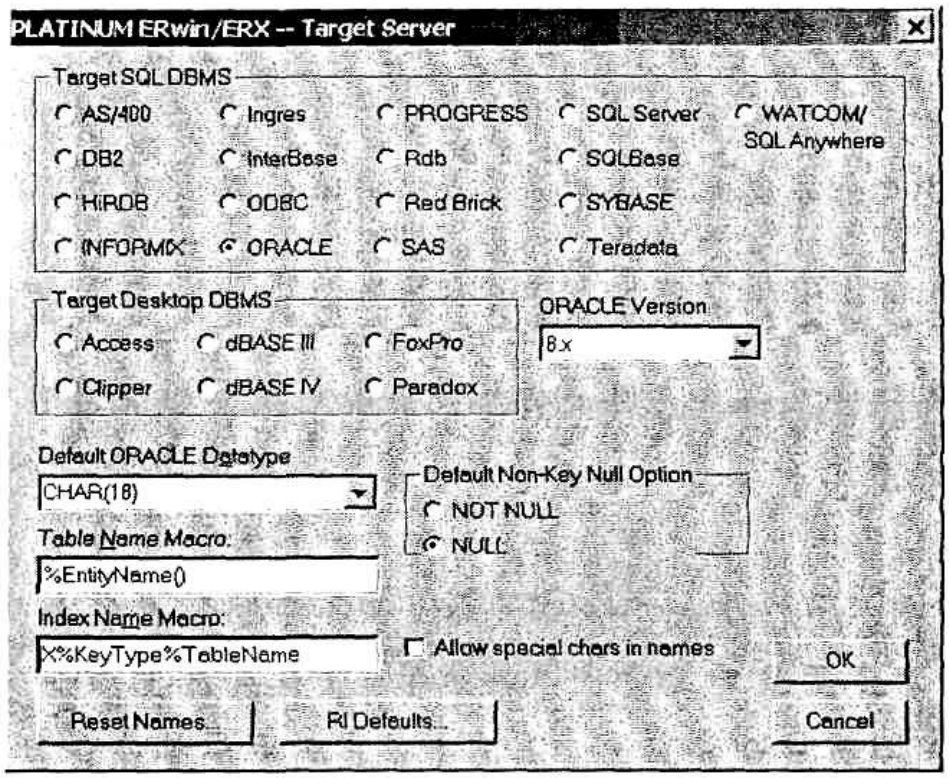


Рис. 3.10. Выбор целевой СУБД

3.4.2. Нотации, используемые при построении физической модели

Так же как и при изображении логической модели, для представления физической модели используется несколько нотаций (рис. 3.11), а именно: IDEF1X (Integration DEFinition for Information Modeling), IE (Information Engineering) и DM (Dimensional Modeling). Как видим, по сравнению с логическим моделированием появилась дополнительная нотация - DM, о которой далее будет сказано особо.

Вид панели инструментов (ERWin Toolbox) для физической модели в нотации IDEF1X представлен на рис. 3.12.

Кнопка  соответствует таблице базы данных, кнопка  представлению (view). Кнопки  , и  несут ту же функциональность, что аналогичные кнопки в логической модели. Кнопка  (view relationship) используется для связывания представлений.

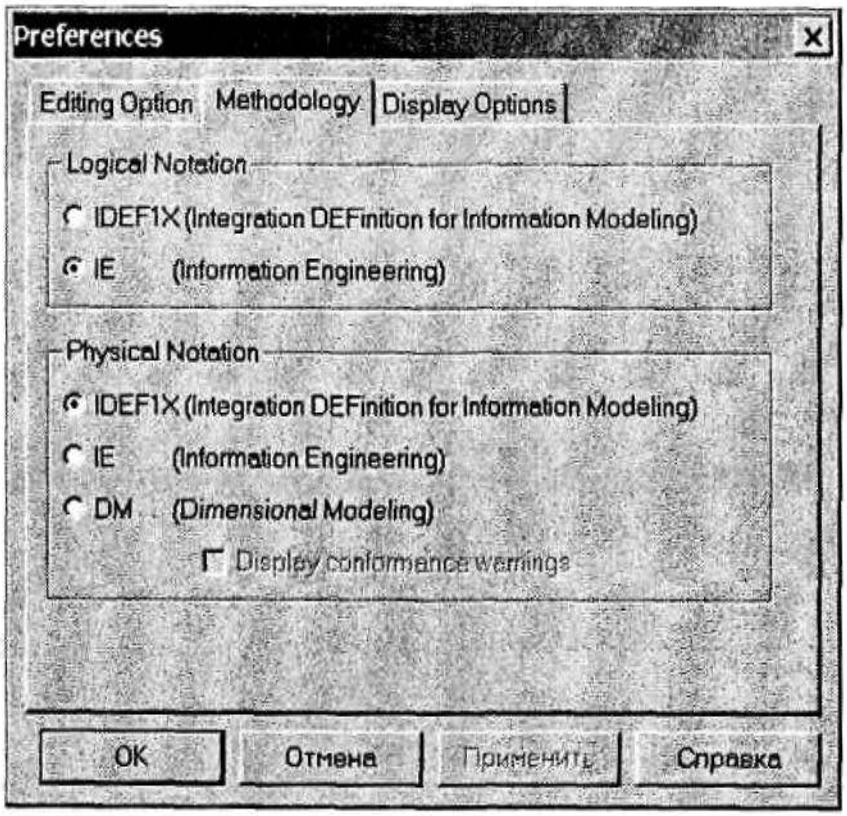


Рис. 3.11. Выбор методологии при создании физической модели



Рис. 3.12. Вид панели инструментов для физической модели
в нотации IDEFIX

На рис. 3.13 изображена физическая модель в нотации IDEF1X полученная из логической модели (*см.* рис. 2.99).

Редактор таблиц на стадии физического моделирования имеет отличия от соответствующего редактора, используемого на стадии логического моделирования. Более того, он имеет специфические особенности, связанные с характеристиками выбранной целевой СУБД. На рис. 3.14 представлен вид окна редактора таблиц **(Table Editor**) для целевой СУБД Access, а на рис. 3.15 - для целевой СУБД Oracle.

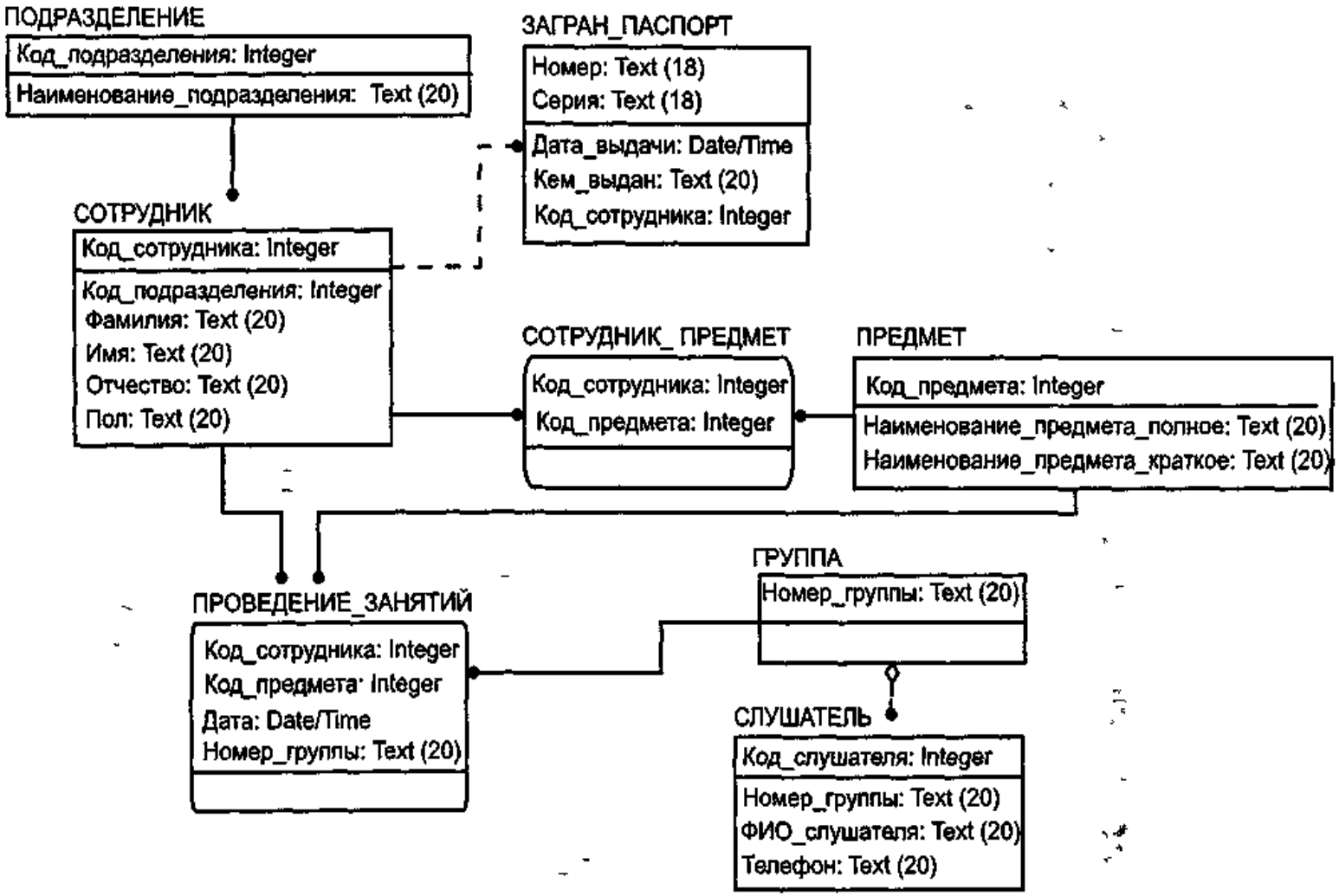


Рис. 3.13. Физическая модель данных

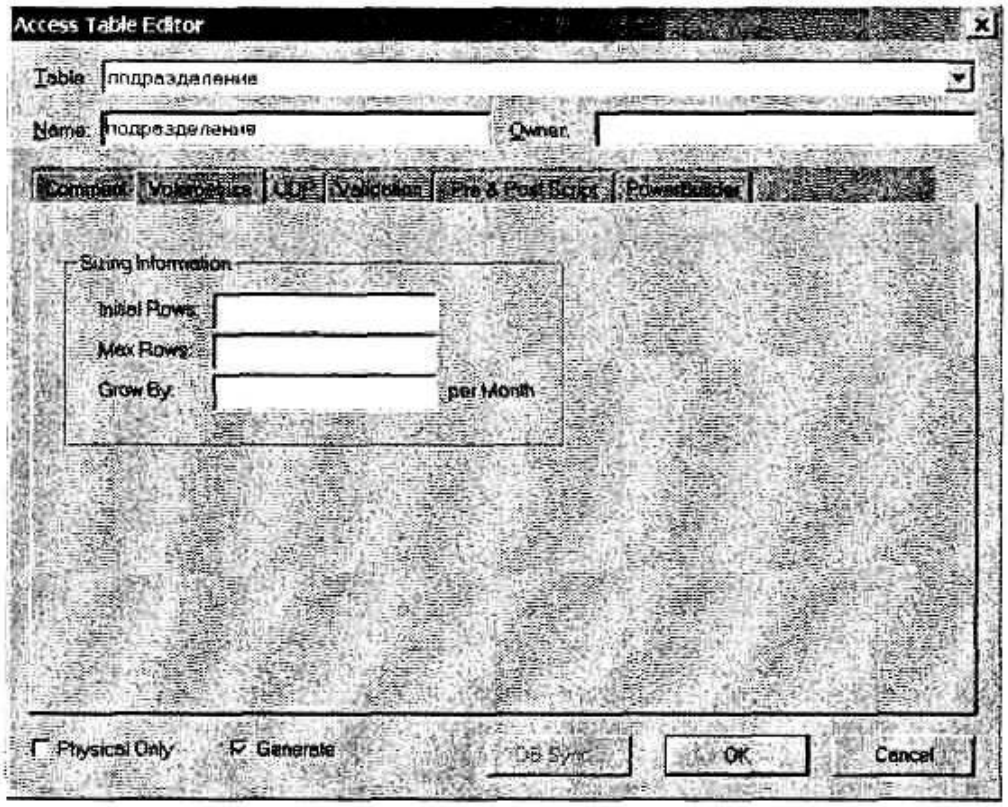


Рис. 3.14. Вид окна редактора таблиц для целевой СУБД Access

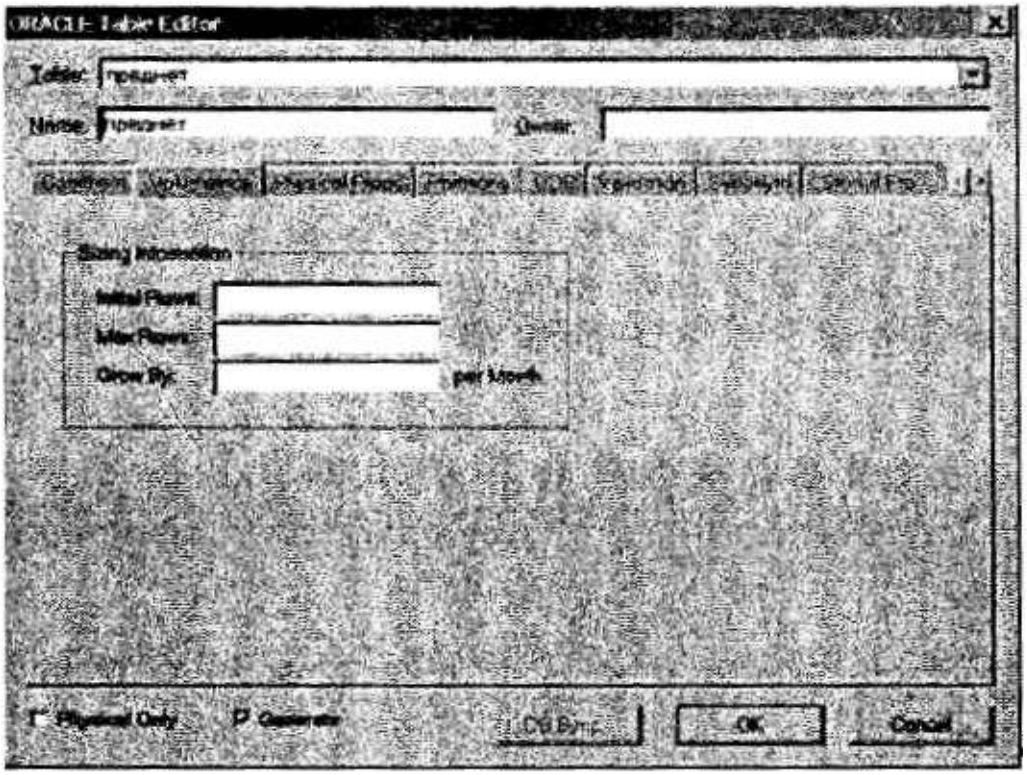


Рис. 3.15. Вид окна редактора таблиц для целевой СУБД Oracle

Контекстные меню также имеют некоторые различия в зависимости от выбранной целевой СУБД (рис. 3.16, 3.17).

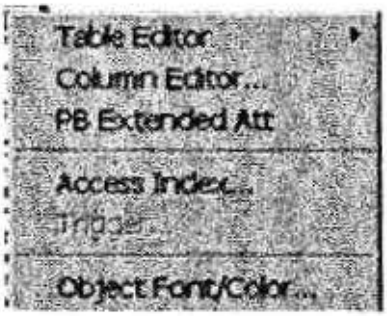


Рис. 3.16. Контекстное меню для целевой СУБД Access

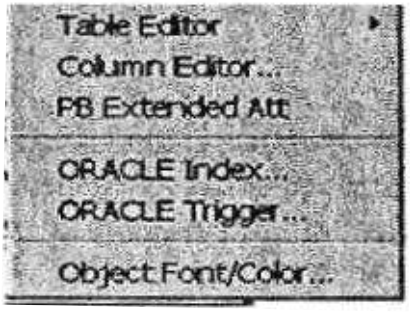


Рис. 3.17. Контекстное меню для целевой СУБД Oracle

Вкладка **Volumetrics** может быть использована для оценки размера БД (рис. 3.18). Для того чтобы оценить размер БД, предварительно для каждой таблицы нужно задать число записей в них.

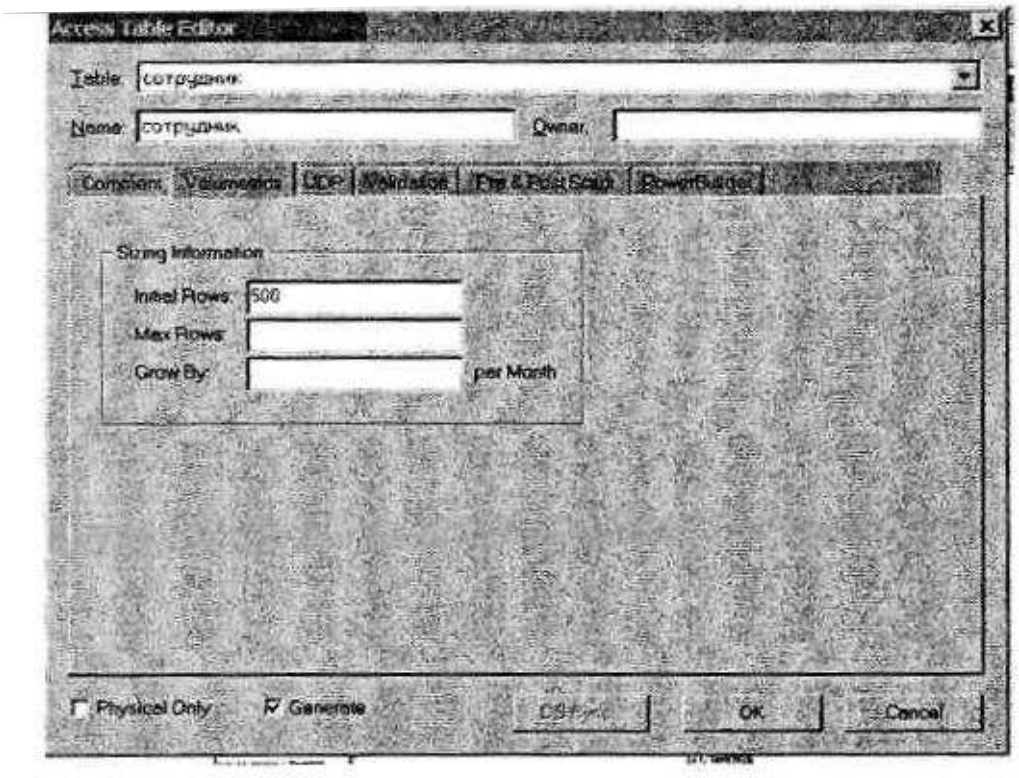


Рис. 3.18. Вид окна редактора таблиц. Вкладка **Volumetrics**

Посмотреть результат вычислений можно, выбрав позиции меню **Tasks/Generate Reports** и затем - нужный отчет. При открытии отчета появится окно **Опций Отчета** (рис. 3.19). Если, описывая размер таблицы, указывать Grow By (прирост числа записей в месяц), то в отчете можно посмотреть не только начальный размер таблицы, но и прогнозируемый размер через заданное число месяцев.

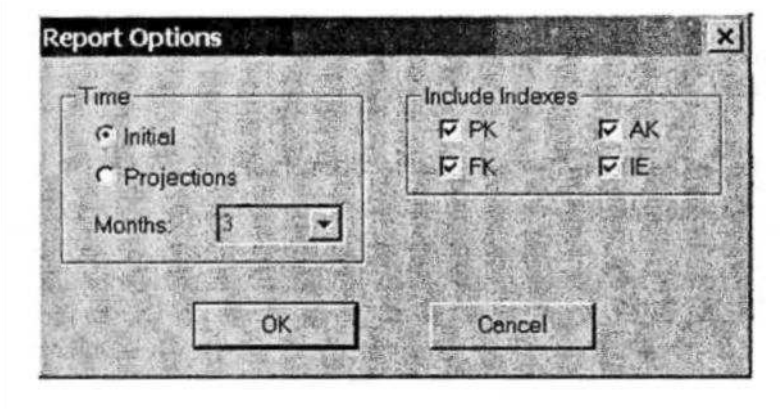


Рис. 3.19. Вид окна **Опции отчета**

3.4.3. Уровни просмотра физической модели

Для физической модели, так же как для логической, существует возможность изменения уровня просмотра модели. Для этого можно воспользоваться одной из изображенных кнопок панели инструментов:  . Первая из них соответствует уровню сущностей, вторая - уровню колонок, третья - уровню определения (comments view). Эти уровни полностью совпадают с описанными при рассмотрении логической модели.

Контекстное меню уровня просмотра для физической модели (рис. 3.20) несколько отличается от соответствующего меню для логической модели. Как мы видим, по сравнению с рис. 2.101 отсутствует позиция меню Icon, но присутствует Physical Order. *Атрибуту* в логической модели в физической модели соответствует термин *Колонка (Column),* а *Сущности (Entity) — Таблица (Table).*

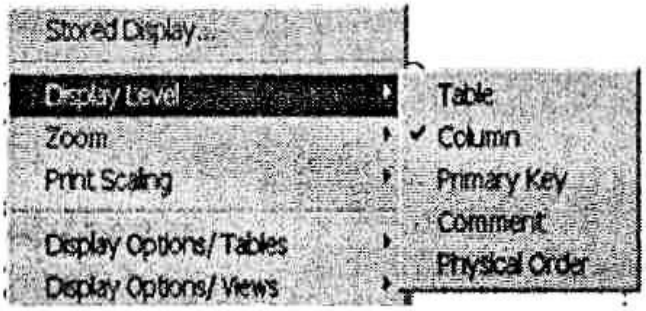


Рис. 3.20. Контекстное меню «Переключения уровней отображения для физической модели»

3.4.4. Сравнение логической и физической моделей

На рис. 3.21 изображен фрагмент логической модели. Сущность ДАТА отмечена как *Logical Only,* поэтому на физической модели соответствующая ей таблица отсутствует (рис. 3.22).

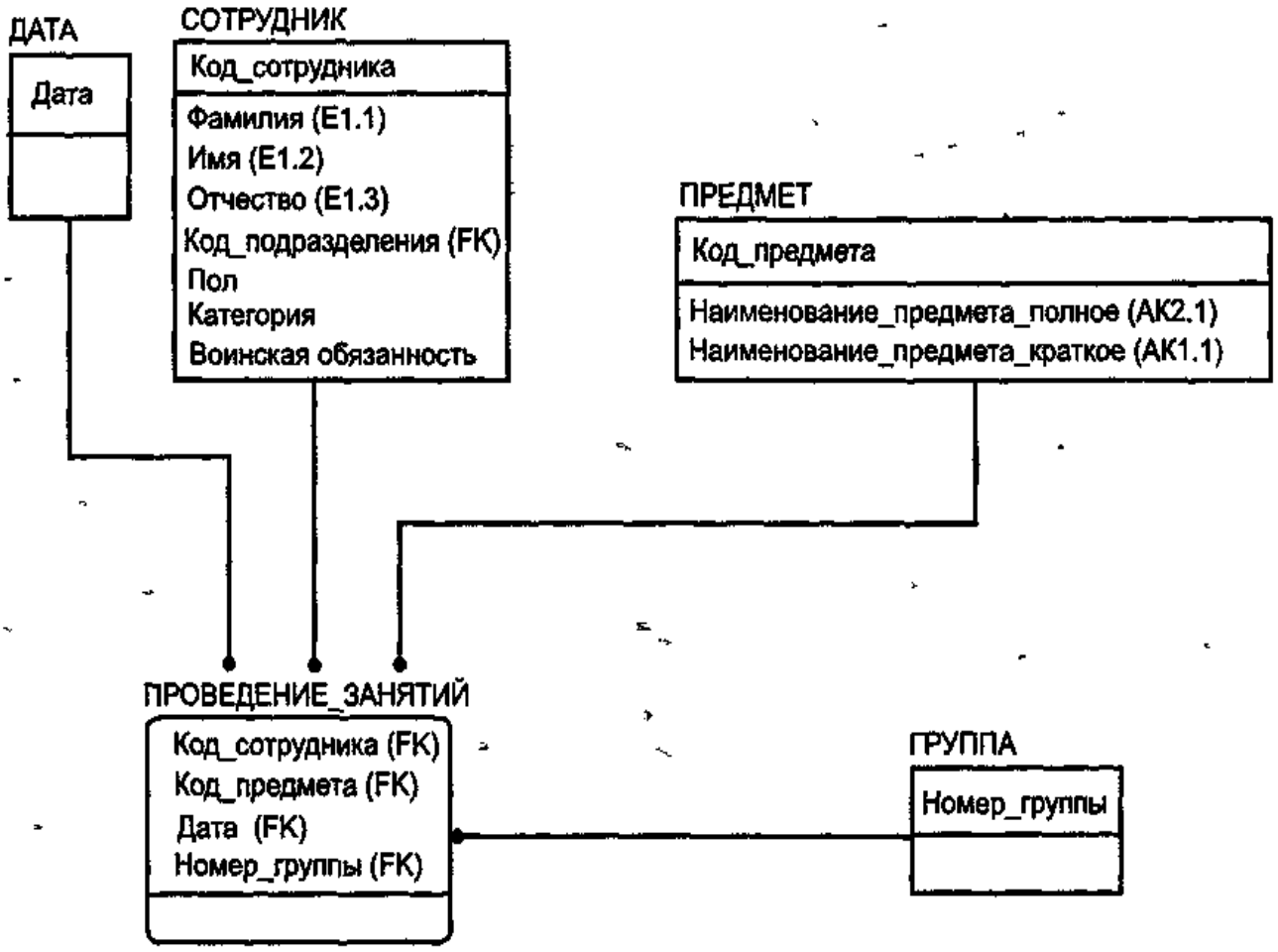


Рис. 3.21. Фрагмент логической модели

На физической модели появилась дополнительная связующая таблица СОТРУДНИК_ПРЕДМЕТ, которая разрывает связь M:M между этими сущностями. Преобразование связи M:M происходит автоматически. В физической модели было создано представление (View), которое показывает фамилии сотрудников и названия предметов, которыми они владеют (V_S4). На схеме физической модели оно изображено пунктирным прямоугольником с закругленными углами, к которому ведут также пунктирные линии. На схеме логической модели View не отражается.

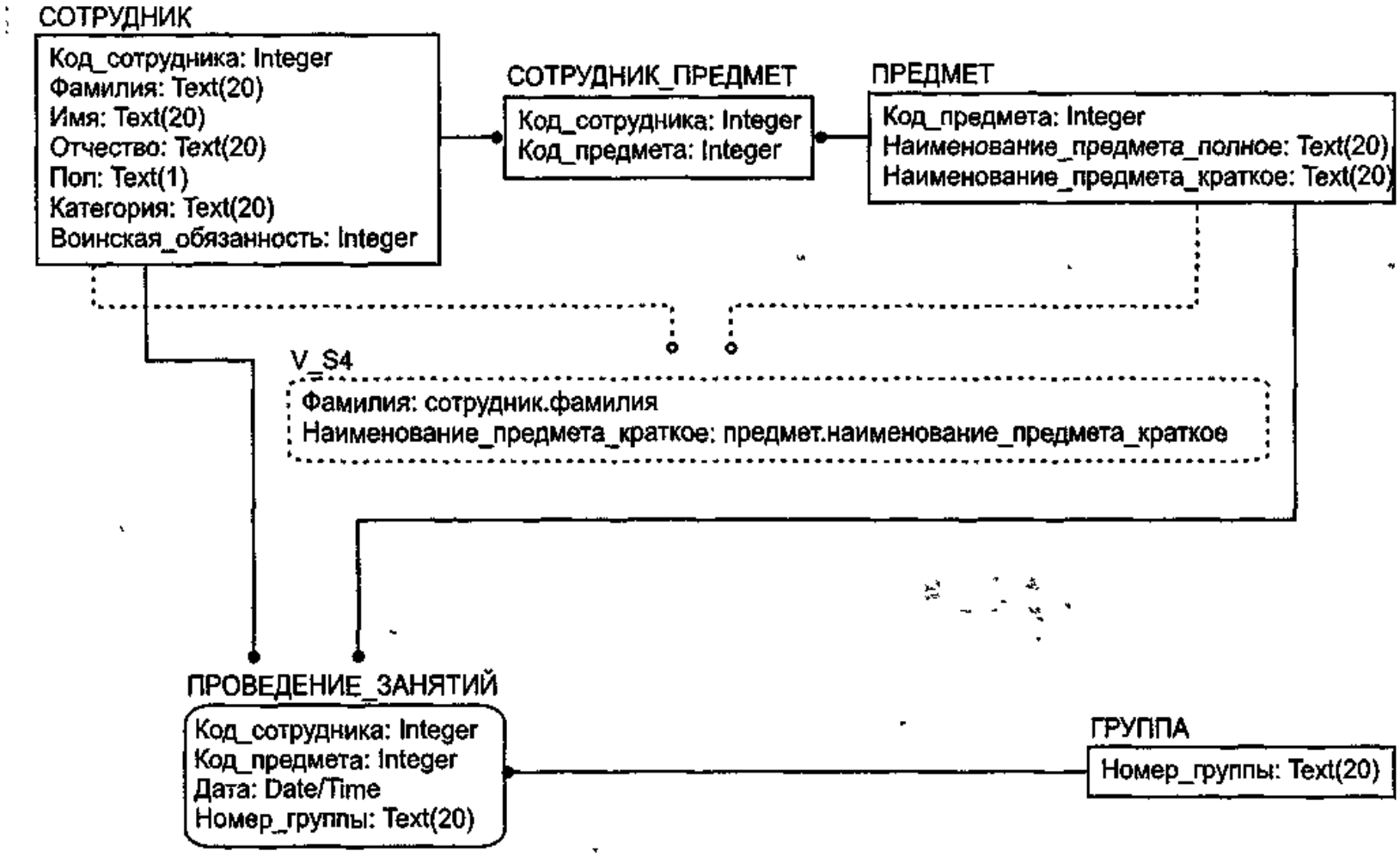


Рис. 3.22. Фрагмент физической модели

3.4.5. Создание хранилищ данных

Хранилища данных имеют специфику как логической структуры, так и физической организации данных и обеспечения доступа к ним.

В хранилищах данных часто используют *многомерные модели данных.* В связи с тем что в хранилищах данных обрабатываются большие объемы данных, предпринимаются меры, обеспечивающие более быстрый доступ к ним. Часто это происходит за счет нарушения нормализации данных, что, в свою очередь, приводит к возникновению специфических проблем.

Для реализации многомерной модели данных в хранилищах данных часто используется схема «звезда». Такая схема обычно содержит одну большую таблицу, называемую таблицей *факта (fact table),* помещенную в «центр» схемы, и меньшие по объему вспомогательные таблицы, называемыми *таблицами измерения (dimension tables),* соединенные с таблицей факта радиальными связями. Схема может также содержать произвольное число *вынесенных* (консольных) *таблиц (outrigger tables),* соединенных с таблицами измерений.

Как отмечалось ранее, инфологическая модель должна содержать не только описание объектов и связей между ними, но и описание запросов пользователей. И если при проектировании логической структуры реляционной базы данных большее значение имеет первый из названных аспектов, то при создании хранилищ данных, напротив, главную роль играют предполагаемые запросы пользователей. Поэтому перед созданием многомерной модели следует проанализировать протекающие бизнес-процессы.

Для получения многомерной* модели данных (Dimensional Modeling) можно в окне **Preferences** в блоке Physical Notation выбрать значение DM (рис. 3.23). Для того чтобы попасть в соответствующее окно, надо осуществить следующий выбор позиций меню: **Option/ Preference/Methodology.**

Перед переходом в нотацию DM будет выдано предупреждающее сообщение (рис. 3.24), говорящее, что если использовать диагональные линии, которые обычно применяются при изображении схемы «звезда», то обратное преобразование линий автоматически будет выполнить невозможно.

Модель (*см.* рис. 3.24) после преобразования ее в нотацию DM будет иметь вид, представленный на рис. 3.25.

---

* В русскоязычной литературе не сложилась устойчивая терминология по хранилищам данных. В [25] *Dimensional Model* переведено как *Размерная модель.* Иногда ее называют *Модель измерений.*

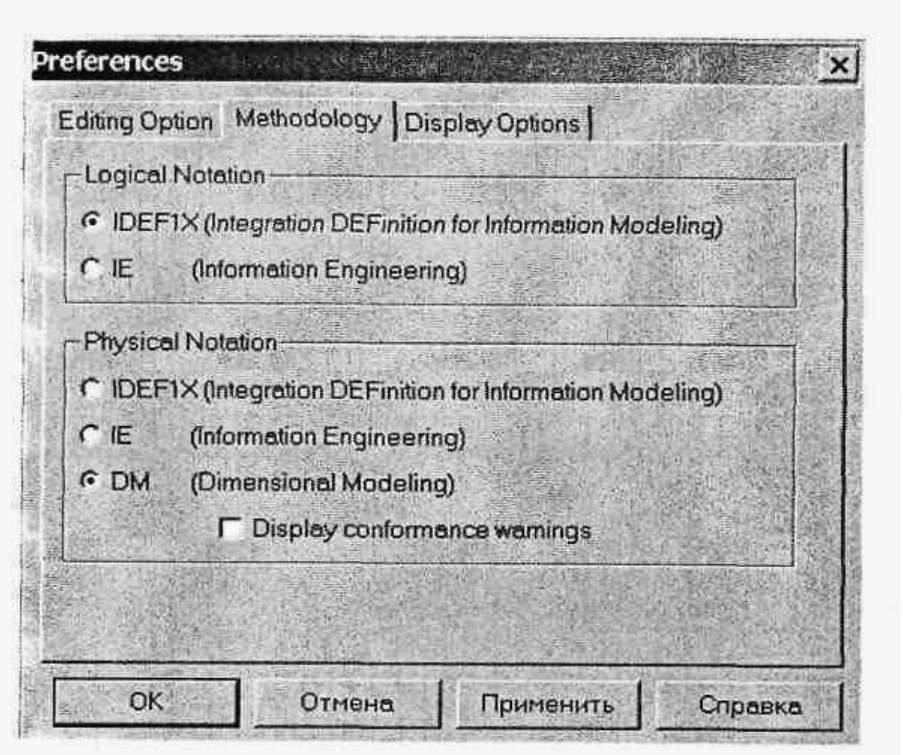


Рис. 3.23. Выбор нотации DM

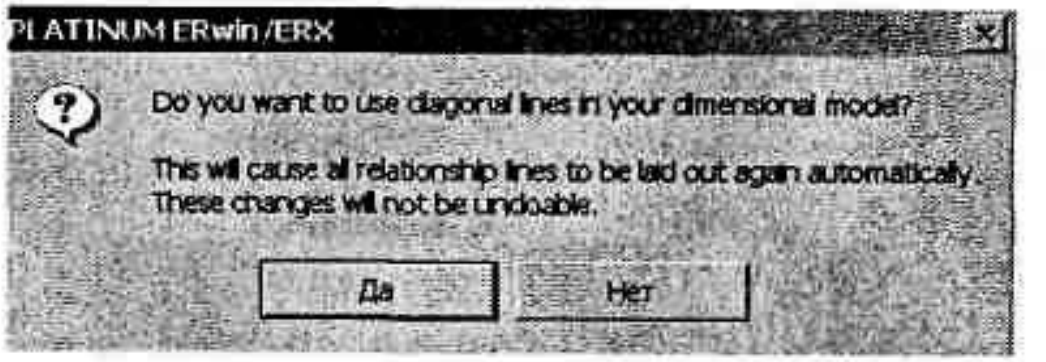


Рис. 3.24. Предупреждающее сообщение при переходе к нотации DM

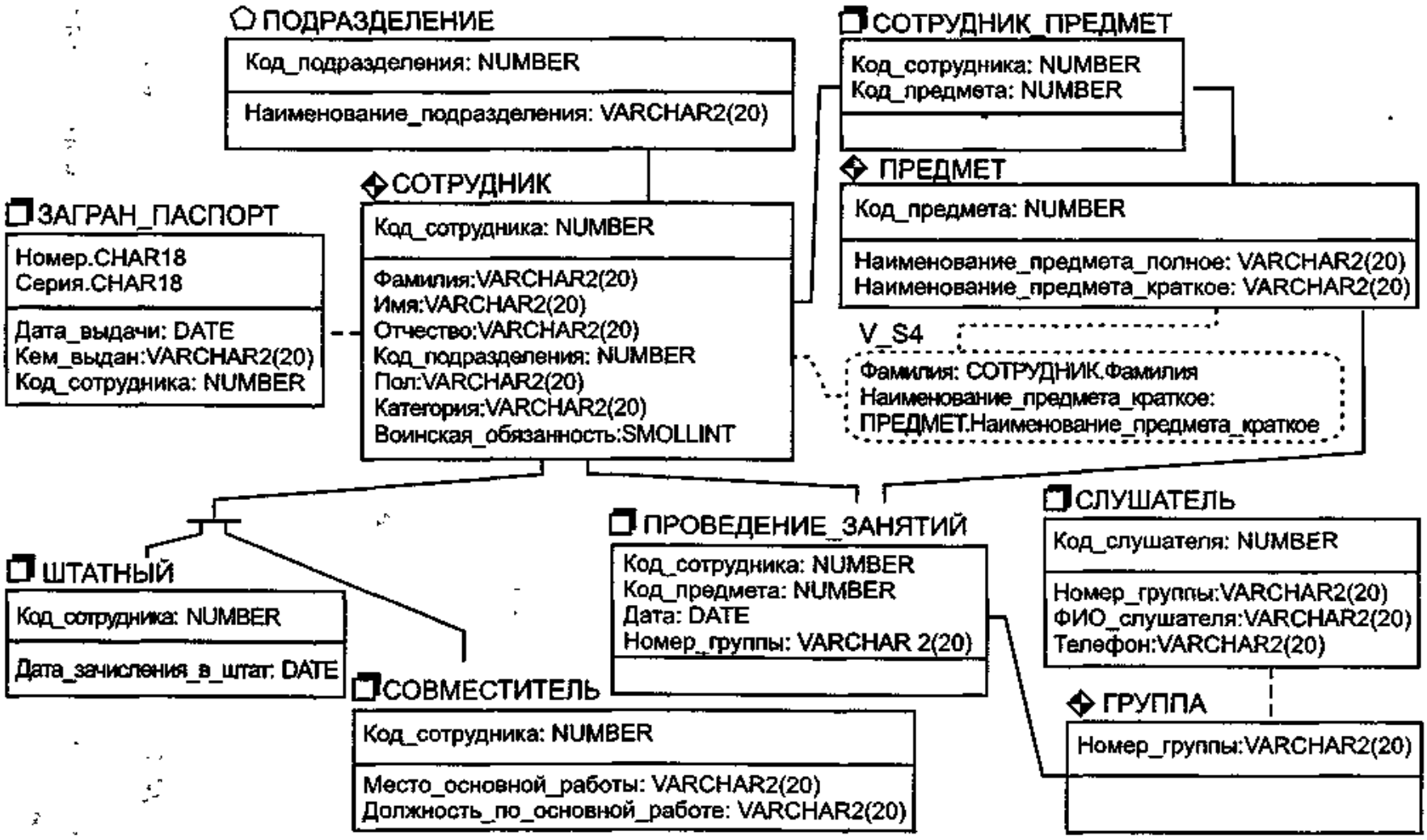


Рис. 3.25. Вид модели в нотации DM

В многомерной модели для различения роли таблиц можно использовать специальные иконки: - для таблицы фактов, - для таблицы измерений, - для консольных таблиц. Чтобы эти иконки появились на схеме, следует выбрать позицию Dimensional Icon ниспадающего контекстного меню (рис. 3.26).

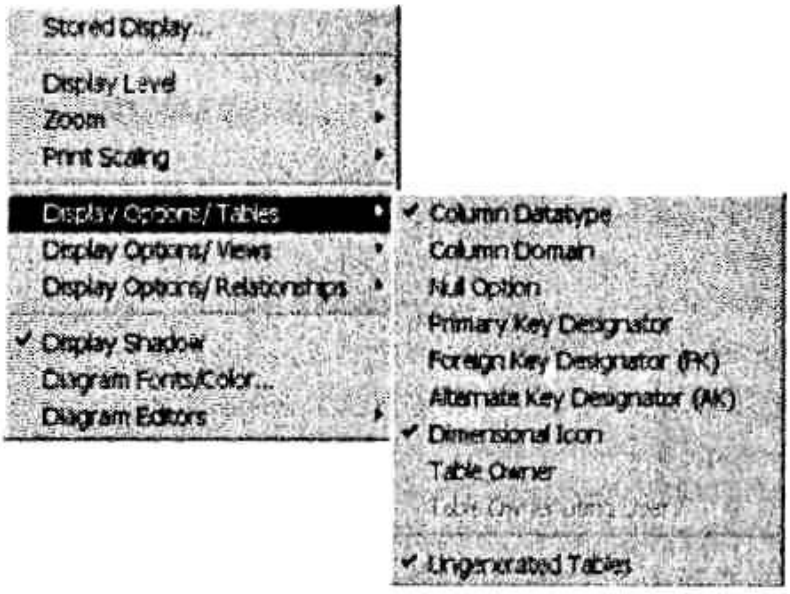


Рис. 3.26. Меню для изображения иконок в DM-модели

При преобразовании физической модели в DM-модель роль каждой таблицы определяется системой автоматически. Если имеется необходимость изменить роль таблиц, то надо в окне **Редактор таблиц** снять флажок **Calculate Automatically** и установить соответствующий переключатель (рис. 3.27).

Верхний ряд кнопок панели инструментов для многомерной модели (рис. 3.28) не отличается от аналогичной панели для физической модели (*см.* рис. 3.12). Отличия заключаются в кнопках, обозначающих тип связи. Кнопка обозначает идентифицирующую связь, кнопка - связь с представлением, а кнопка - неидентифицирующую связь. Как мы видим, в многомерной модели явно не фиксируются направление связи и кардинальное число. Предопределено, что связь идет по направлению от таблицы размерности к таблице факта и от консольных таблиц к таблице размерности.

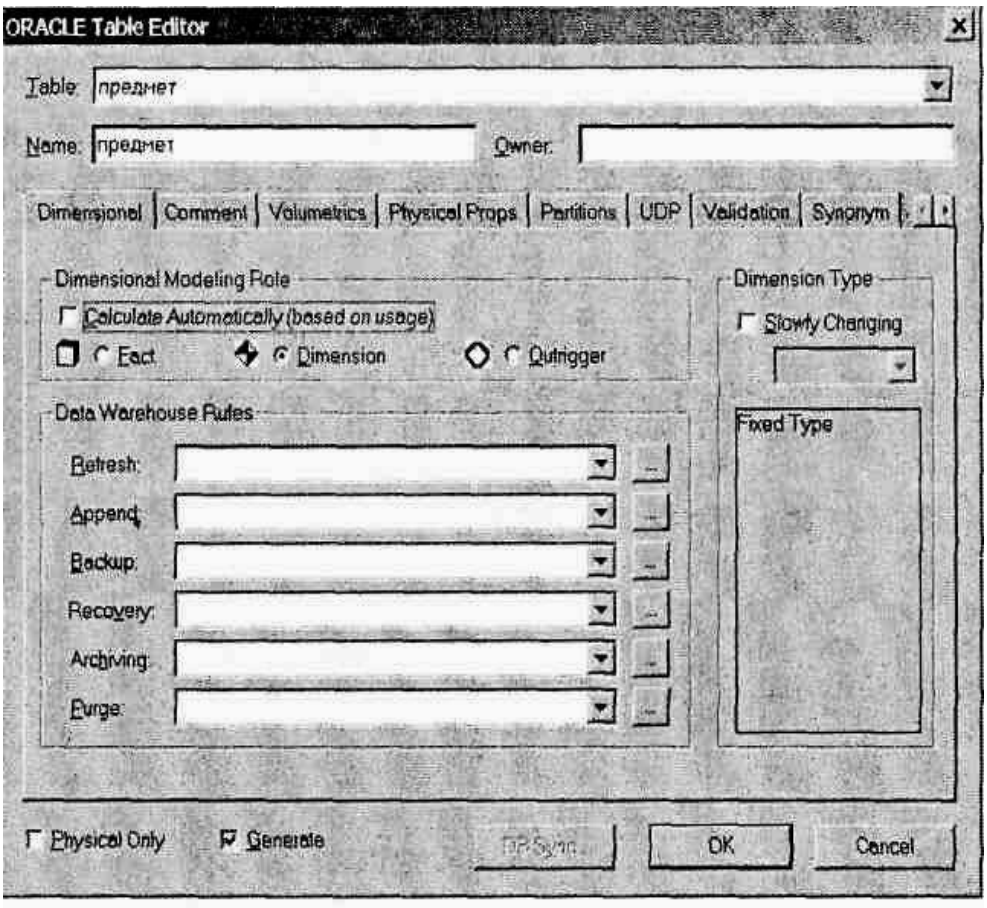


Рис. 3.27. Вид окна **Редактор таблиц** при работе с DM-моделью

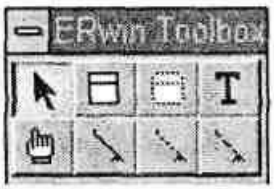


Рис. 3.28. Вид панели инструментов
для многомерной модели

3.4.6. Переход к даталогической модели

На основе физической модели ERWin можно сгенерировать системный каталог СУБД или соответствующий SQL-скрипт. Создание схемы базы данных из концептуальной модели называется прямым проектированием (Forward Engineering). Для генерации системного каталога БД следует выбрать пункт меню **Tasks/Forward Engineer/Schema Generation** или щелкнуть по кнопке  на панели инструментов. В результате появится окно (рис. 3.29) генерации схемы в выбранной целевой СУБД.

Кнопка Filter позволяет убрать объекты из схемы. Кнопка Preview позволяет просмотреть SQL-скрипт, создаваемый ERWin для генерации системного каталога СУБД. С помощью кнопки Print можно вывести на печать

созданный SQL-скрипт. Кнопка Report сохранит тот же скрипт в текстовом файле. Кнопка Generate запускает процесс генерации схемы.

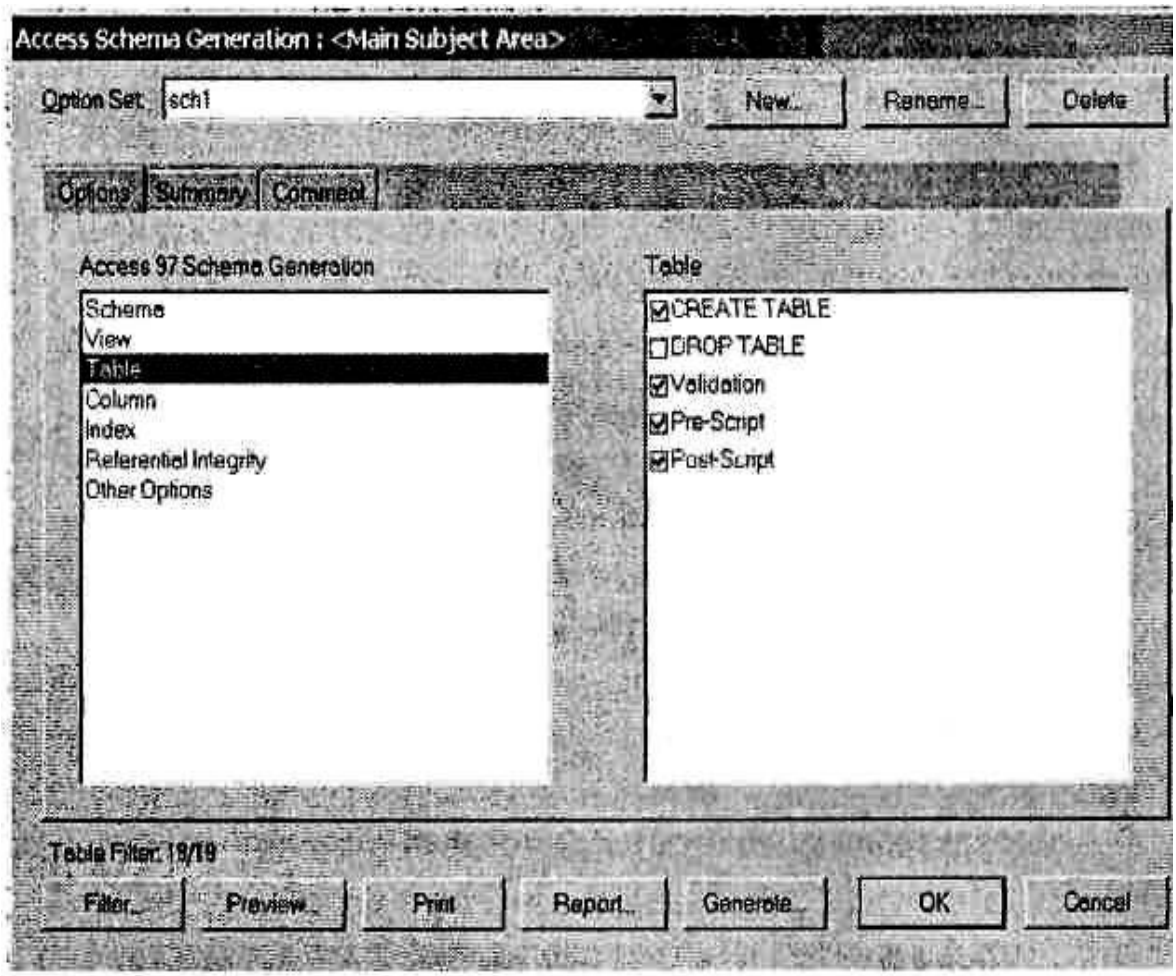


Рис. 3.29. Вид окна генерации схемы

На это следует обратить внимание

1. На построение логической модели базы данных оказывает влияние множество факторов, и все они в комплексе должны быть учтены при создании даталогической модели.

2. Подход к проектированию логической структуры базы данных, рассмотренный для реляционной модели, может быть применен и при проектировании других структурированных баз данных. При этом в алгоритме перехода от ER-модели к модели, поддерживаемой конкретной СУБД, должны быть учтены особенности модели данных целевой СУБД.

3. Поскольку базовая ER-модель является объектной по своей сути, то она может быть использована и при создании объектных баз данных.

4. Использование CASE-средств позволяет улучшить качество разрабатываемых проектов, а при создании крупных корпоративных систем является практически неизбежным.

5. Применение CASE-средств не освобождает проектировщика от понимания не только общей сущности, но и деталей даталогического проектирования.

Контрольные вопросы

1. Что называется даталогическим проектированием?
2. Какая информация является исходной для даталогического проектирования?
3. В чем заключается проектирование логической структуры базы данных для каждого из известных вам классов СУБД или конкретных СУБД?
4. Какие критерии используются для оценки спроектированной базы данных?
5. В каких случаях и для обеспечения каких целей вводятся искусственные идентификаторы?
6. Как отображаются простой объект и его единичные свойства в реляционной базе данных? в других известных вам СУБД?
7. Как отображаются условные свойства объектов в реляционной базе данных? в других известных вам СУБД?
8. Как отображаются множественные свойства объектов в реляционной базе данных? в других известных вам СУБД?
9. Как отображается отношение типа 1:1 между объектами в реляционной базе данных? в других известных вам СУБД? Влияет ли при этом класс принадлежности объектов на число требуемых файлов/таблиц?
10. Как отображается отношение типа 1:M между объектами в реляционной базе данных? в других известных вам СУБД? Влияет ли при этом класс принадлежности объектов на число требуемых файлов/ таблиц?
11. Как отображается отношение типа M:M между объектами в реляционной базе данных? в других известных вам СУБД? Влияет ли при этом класс принадлежности объектов на число требуемых файлов/таблиц?
12. Как отображается в реляционной модели составной объект? в других известных вам СУБД?
13. Как отображается в реляционной модели обобщенный объект?
14. Как отображается в реляционной модели агрегированный объект?
15. Все ли показатели, отображенные в инфологической модели, должны включаться в базу данных?
16. Какие факторы влияют на принятие решения о выборе показателей, хранимых в базе данных?
17. В каком случае надо проводить вертикальное и горизонтальное разбиение файлов/таблиц базы данных?
18. Что в ERWin называется физической моделью?
19. Как можно построить физическую модель в ERWin?
20. В чем состоят отличия физической модели от логической (в ERWin)?

21. Может ли физическая модель содержать элементы (таблицы, поля и др.), которые отсутствовали в логической модели (в ERWin)? Если да, то чем это может быть вызвано?

22. Может ли логическая модель содержать элементы (сущности, атрибуты), которые не переносятся в физическую модель (в ERWin)? Если да, то чем это может быть вызвано?

23. Что называется «целевой СУБД»? Как можно выбрать целевую СУБД?

24. К каким изменениям в физической модели приведет смена целевой СУБД?

25. В каких нотациях может быть построена физическая модель? Чем они отличаются друг от друга?

26. Как можно построить хранилище данных с использованием ERWin?

27. Что представляет собой схема «звезда»?

28. Что в хранилище данных называется таблицей факта? таблицами измерения? «вынесенными» (консольными) таблицами?

Глава 4 ЦЕЛОСТНОСТЬ БАЗЫ ДАННЫХ

4.1. Классификация ограничений целостности

Обеспечение целостности данных является важнейшей задачей при проектировании и эксплуатации систем обработки данных (СОД).

«Проблема *целостности* состоит в обеспечении ... правильности данных в базе данных в любой момент времени» [14]. *Целостность* — актуальность и непротиворечивость информации, ее защищенность от разрушения и несанкционированного изменения.

Целостность является одним из аспектов информационной безопасности наряду с доступностью - возможностью с приемлемыми затратами получить требуемую информационную услугу, и конфиденциальностью - защитой от несанкционированного прочтения.

Целостность данных - неотъемлемое свойство базы данных, и ее обеспечение является важнейшей задачей проектирования БнД. Целостность данных описывается набором специальных предложений, называемых *ограничениями целостности.* Ограничения целостности представляют собой утверждения о допустимых значениях отдельных информационных единиц и связях между ними. Эти ограничения определяются в большинстве случаев особенностями предметной области, хотя могут отражать и чисто информационные (лингвистические) характеристики. Например, если используются цифровые коды для обозначения какой-либо номенклатуры, то ограничения на тип используемых символов для соответствующего атрибута в БД определяются не спецификой предметной области, а просто выбранным способом кодирования, а ограничение, выражающееся в том, что возраст работающего должен быть не менее 16 лет, — трудовым законодательством, т.е. только спецификой предметной области.

При выполнении операций над БД проверяется выполнение ограничений целостности. Действия, приводящие к нарушению подобных ограничений, отвергаются.

Ограничения целостности могут классифицироваться по разным признакам (рис. 4.1).

Ограничения целостности могут относиться к разным информационным объектам[*]: атрибутам (полям), кортежам (строкам, записям), отношениям (таблицам, файлам)[**], связям между файлами и т.п.

---

[*] Для простоты изложения будем далее пользоваться в основном терминологией реляционных СУБД. Однако все сказанное ниже относится и к СУБД других классов.

[**] Поскольку в разных СУБД используется разная терминология, а изложение материала не привязано к какой-то конкретной СУБД, то все эти термины здесь используются как синонимы, хотя на самом деле это не так.

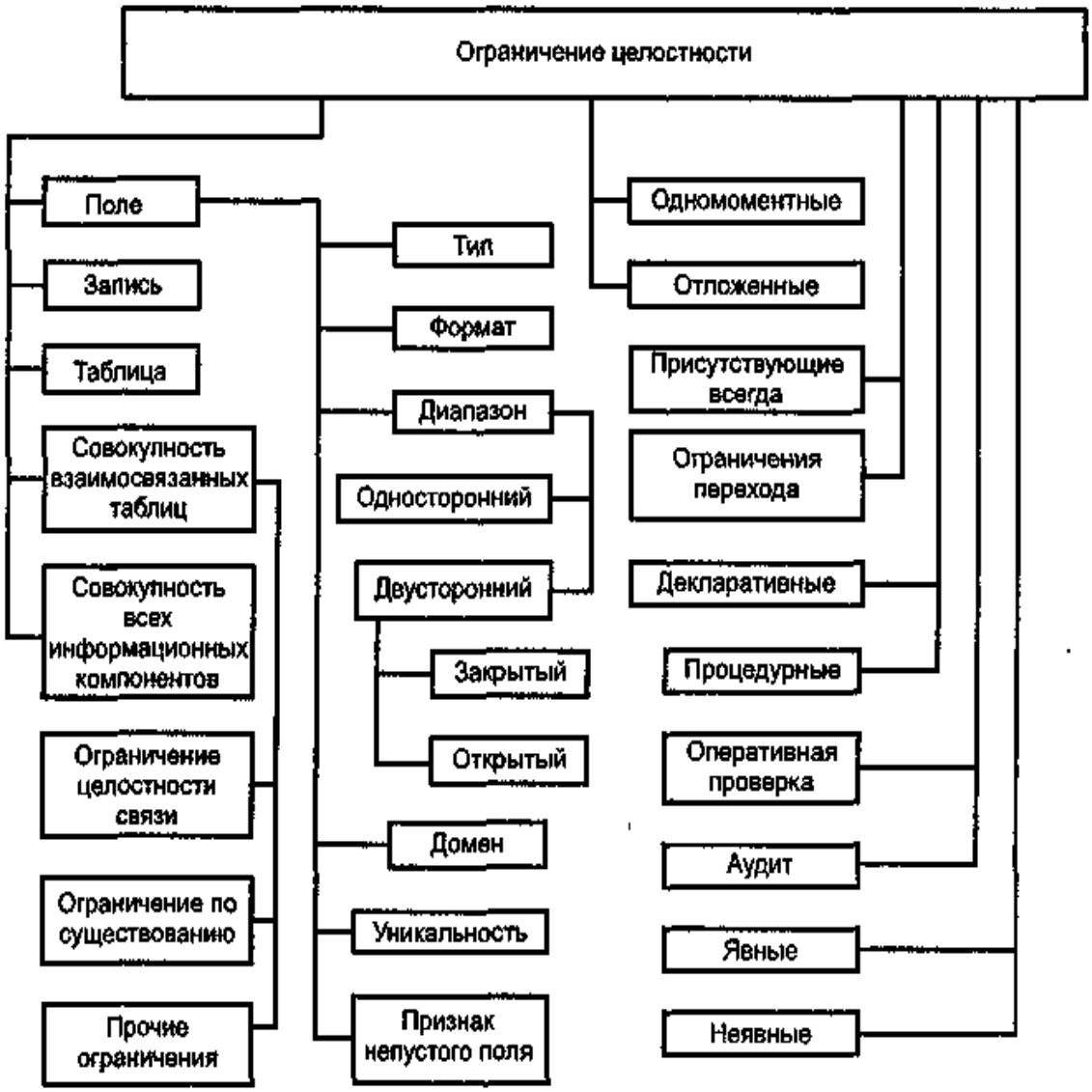


Рис. 4.1. Общая схема классификации ограничений целостности

1. ***Поле.*** Для него чаще всего используются следующие виды ограничений.

1.1. *Тип и формат поля.* Тип поля определяет допустимые для данного поля символы, а иногда и более жесткие ограничения на допустимые значения (как, например, для полей типа *дата* или *логическое).*

1.2. *Задание диапазона значений.* Обычно используется для числовых полей.

1.2.1. Различают *односторонние* и *двусторонние* диапазоны. Первые фиксируют значение только одной из границ (верхней или нижней), вторые - обеих границ. Так, например, до определенного времени в нашей стране ограничивался как нижний, так и верхний предел заработной платы. Это пример двустороннего закрытого диапазона. Затем ограничение по верхнему пределу было снято: заработная плата не может быть меньше установленного

минимума, но максимальное ее значение законодательно не определено — ограничение стало односторонним.

1.2.2. Диапазоны бывают *открытые* и *закрытые.* Односторонний диапазон всегда является открытым, двусторонний может быть как открытым, так и закрытым.

Двусторонний диапазон будет открытым, если допустимые значения меньше «левой» границы и больше «правой» (рис. 4.2). Задание двусторонних открытых диапазонов используется гораздо реже, чем закрытых. Некоторые СУБД поддерживают высокоуровневые средства задания двусторонних закрытых диапазонов и не поддерживают - открытых. Пример открытого диапазона: орган социального обеспечения поддерживает базу данных, содержащих записи о людях моложе 16 лет или старше 60.

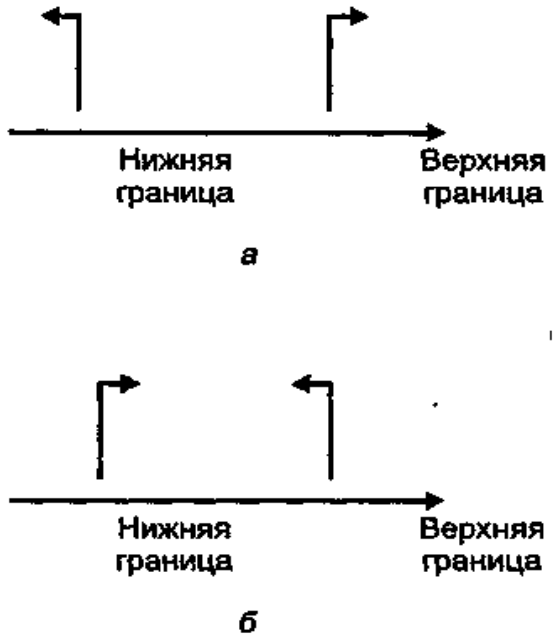


Рис. 4.2. Графическая иллюстрация понятий открытого *(а)* и закрытого *(б)* двустороннего диапазона

1.3. *Признак непустого поля.* Характеризует недопустимость пустого значения поля в БД. Так, например, в таблице, содержащей сведения о сотрудниках, поля «Фамилия», «Имя», «Отчество», «Оклад» должны обязательно иметь какое-то значение, а у поля «Ученая_степень» значение может отсутствовать.

1.4. *Задание домена.* Поле может принимать значение из заданного множества. Множество возможных значений какого-либо атрибута называется *доменом.* Домен может задаваться перечислением входящих в него значений (например, значением поля «Пол» может быть только либо «мужской» либо «женский»; значением поля «Должность» для профессорско-преподавательского состава может быть: «ассистент», «старший преподаватель», «доцент» и «профессор») или алгоритмом вычисления допустимых значений (как это обычно происходит для полей типа «Дата»). Последний из

приведенных примеров свидетельствует не только о возможностях СУБД по поддержанию целостности данных, но и о важности процедуры выбора типа данных при проектировании баз данных.

Следует обратить внимание, что термин «домен» здесь используется не как сугубо реляционный, а для обозначения множества возможных значений какого-либо атрибута безотносительно к используемой модели данных.

1.5. Специфическим ограничением на значение поля является *признак* его *уникальности.* Это ограничение проверяет допустимость значения данного поля, но при этом просматривается вся таблица (файл). Поэтому, с одной стороны, данное ограничение правильнее было бы отнести к ограничениям на таблицу. Но, с другой стороны, ограничение на уникальность поля проверяется сразу после ввода значения конкретного поля, в отличие от большинства других ограничений целостности на таблицу.

Признак уникальности поля тесно связан с понятием *ключа,* но уже последнего, поскольку ключ может быть представлен не только одним полем, а совокупностью полей (составной ключ). Уникальное поле является вероятным ключом данного отношения. При наличии нескольких вероятных ключей один из них должен быть выбран в качестве первичного ключа. Поле, выбранное в качестве первичного ключа, не должно иметь пустых значений. Не все СУБД поддерживают концепцию ключа, т.е. позволяют определять ключ при описании БД. Некоторые СУБД для каждого файла (таблицы) требуют обязательно определять ключ при описании базы данных. Другие СУБД (например, Access), в принципе поддерживая концепцию ключа, разрешают создавать таблицы, в которых ключ не задан.

Ограничение на уникальность чаще всего возникает при отображении в базе данных каких-то объектов, и уникальное поле является идентификатором этого объекта. Поэтому это ограничение целостности иногда называется *ограничением целостности объекта (сущности).*

1.6. Очень важным видом ограничений целостности являются *функциональные зависимости.* Информацию об имеющих место в данной предметной области функциональных зависимостях можно извлечь из инфологической модели (*см.* разд. 4.2). Эта информация используется и при проектировании базы данных, и для контроля целостности при ее функционировании. Если БД спроектирована правильно, т.е. она находится в 4-й нормальной форме, то, определяя ключи и вероятные ключи отношений, тем самым определяются и имеющиеся функциональные зависимости между атрибутами.

1.7. Рассмотренные выше ограничения определяли проверки значения поля вне зависимости от того, вводится это значение впервые или корректируются имеющиеся в базе данных значения. Ограничения, которые используются только при проверке допустимости корректировки, называют *ограничениями перехода* (или *динамическими ограничениями*). Например, если в базе данных имеются поля «Возраст_сотрудника», «Стаж_работы» и т.п., то при корректировке значения этих полей могут только увеличиваться. В аспекте правильности проектирования БД приведенные выше для иллюстрации поля, особенно поле «Возраст_сотрудника», лучше вообще не хранить в базе

данных, а получать расчетным путем. Это не только существенно упростит ведение базы данных, но и облегчит процесс обеспечения целостности данных.

Другим примером ограничения перехода является корректировка поля «Семейное_положение». Так, значение «вдовец» может быть исправлено только на «женат», а «холост» не может быть исправлен на «разведен» и т.п.

Ограничение целостности может относиться как к реальному, так и к виртуальному полю, т.е. полю, которое в явном виде в таблице не хранится (например, если в БД фиксируется только поле «Дата__рождения», а ограничение накладывается на возраст). Последнее порождает дополнительные сложности. Так, в рассматриваемом примере значение свойства «возраст» в явном виде в БД не хранится - значит, надо разрабатывать специальную процедуру проверки соблюдения ограничения в каждый момент времени и решать вопрос о периодичности использования этой процедуры для проверки БД. Когда проверка происходит в момент ввода данных, то она играет двоякую роль: контроль правильности ввода данных и соответствие человека, о котором вводятся данные, предъявляемым к нему требованиям. При периодической проверке данного ограничения после ввода данных происходит проверка соответствия самой предметной области установленным требованиям.

Проверка соблюдения выполнения ограничений целостности, естественно, замедляет процесс обработки данных. Поэтому иногда бывает полезным определить, могут ли внесенные изменения базы данных привести к нарушению тех или иных ограничений целостности, и проводить контроль только в случае, когда такие нарушения потенциально могут возникнуть. Например, в предметной области может быть ограничение, заключающееся в том, что зарплата начальника не может быть ниже зарплаты его подчиненных. Тогда в случае увеличения зарплаты начальника соблюдение этого ограничения проверять не надо, а в случае уменьшения - надо.

2. ***Кортеж* (*запись, строка*)**. Здесь имеются в виду ограничения на соотношения значений отдельных полей в пределах одной строки. В качестве ограничения на соотношения полей внутри одного кортежа можно привести следующее: значение поля «Стаж» не должно превышать [«Возраст» - 16] (предполагается, что трудовой стаж человека начинается не ранее чем в 16 лет).

3. ***Таблица.*** Ограничения целостности, относящиеся к таблице, затрагивают разные ее строки. Чаще всего для проверки их соблюдения приходится просматривать всю таблицу. К таким ограничениям относятся рассмотренные выше ограничения на уникальность поля или совокупности полей. Другим примером может служить ограничение для таблицы, содержащей сведения об окладах сотрудников, заключающееся в том, что средняя зарплата сотрудников не должна превосходить заданную величину. Такое ограничение может иметь место в случае, когда фонд заработной платы формируется исходя из величины нормативной средней заработной платы одного сотрудника и численности сотрудников.

Имеется ряд ограничений целостности, которые проверяют соотношения между записями одной таблицы: 1) нельзя быть родителем и ребенком одного и того же человека; 2) год рождения родителя должен быть меньше, чем год рождения ребенка. Первый из приведенных примеров является частным случаем более общего ограничения на отсутствие циклов. К аналогичным ограничениям относятся ограничения на наличие циклов при определении состава изделия (узел не может входить сам в себя), при описании организационной структуры и во многих других случаях. Если СУБД не позволяет контролировать подобные ограничения целостности, то следует написать универсальную программу (создать процедуру), позволяющую это делать, поскольку такого рода проверки нужны достаточно часто.

4. ***Совокупность взаимосвязанных таблиц***. Эти ограничения отличаются от рассмотренных выше, которые затрагивали информационные единицы в пределах одной таблицы.

4.1. Наиболее часто встречающееся из этих ограничений - *ограничение целостности связи.* Оно выражается в том, что значение атрибута, отражающего связь между объектами и являющегося внешним ключом отношения, обязательно должно совпадать с одним из значений атрибута, являющегося ключом отношения, описывающего соответствующий объект. Например, если в базе данных существует таблица, отражающая связь между преподавателями и дисциплинами, которые каждый из них может преподавать, то код преподавателя в этой таблице должен соответствовать одному из кодов в таблице «Преподаватели», а код дисциплины - значению соответствующего поля в таблице «Дисциплины».

Ограничение целостности связи схоже по смыслу с ограничением путем задания домена, только роль домена здесь играет соответствующая колонка связанного отношения.

4.2. Разновидностью ограничения целостности связи является *ограничение по существованию,* заключающееся в том, что для существования объекта в отношении S_1 необходимо, чтобы он был связан с объектом в отношении S_2.

Например, если сотрудника принимают на работу, то он должен быть «приписан» к какому-либо отделу, т.е. экземпляр записи «Сотрудник» может существовать только при существовании отдела, в котором он работает, и эта связь должна быть обязательно задана. В принципе в предметной области может быть и иная ситуация, допускающая наличие сотрудников, не приписанных ни к какому отделу. В последнем случае ограничение между таблицами «Сотрудник» и «Отдел» будет ограничением по связи, но не будет ограничением по существованию; т.е. ограничение по существованию является более сильным, чем простое ограничение по связи, и предполагает не только наличие соответствующего значения идентификатора отдела в таблице «Отдел», но и недопустимость пустого значения поля «Отдел» в таблице «Сотрудник».

Проблема обеспечения целостности по связи (и по существованию) поразному выглядит при вставке новой записи в связанную таблицу, при кор-

ректировке записей, при удалении связанных записей. Например, если удаляется запись из основной таблицы, то возможны следующие варианты:

- запись в основной таблице можно удалять только в том случае, если нет связанных с ней записей в подчиненной таблице;
- при удалении записи основной таблицы удаляются все связанные с ней записи в зависимой таблице (так называемое каскадное удаление);
- если это просто ограничение по связи, а не ограничение по существованию, то возможен вариант, когда при удалении записи основной таблицы значения соответствующего поля связанных с ней записей должны быть заменены на «пусто».

Удаление записей зависимой таблицы не может привести к нарушению ограничения целостности по связи или существованию.

В предметной области в принципе может быть ситуация «обратной связи по существованию», когда не может существовать запись основной таблицы без связанных с ней зависимых записей. Например, может быть ограничение, декларирующее, что не может быть отдела без сотрудников. Подобного рода ограничения контролировать сложнее, поскольку при удалении каждой записи зависимой таблицы надо смотреть, есть ли еще записи в зависимой таблице с таким же значением внешнего ключа.

При изменении значений полей связи важно, с какой стороны связи изменяется поле. Если изменяется ключевое поле в основной таблице, то чаще всего проводится каскадное изменение соответствующих ему значений в зависимой таблице. Если такое изменение разрешить нельзя, то задача отслеживания изменений и обеспечения целостности БД становится достаточно сложной.

Если изменение касается поля связи в зависимом файле, то при изменении нужно смотреть, есть ли новое значение в соответствующем поле основного файла. Иногда изменения поля связи в зависимом файле должны быть запрещены. Например, если имеется пара связанных таблиц «Отдел»-«Сотрудник», то изменение значения поля «Код отдела» в таблице «Сотрудник» будет означать перевод сотрудника в другой отдел (при изменении значения поля нужно проверять, что скорректированное значение не нарушает целостность по связи). Если же имеется пара связанных таблиц «Студент»-«Успеваемость», то изменение значение поля «Код_студента» в таблице «Успеваемость» следует запретить, поскольку такая корректировка означала бы, что результаты экзамена одного студента будут приписаны другому студенту, что бессмысленно.

В конкретных предметных областях могут быть и более сложные ограничения связи, затрагивающие кардинальность связи (т.е. число элементов в связи). Это можно проиллюстрировать на примере спортивных команд, которые имеют ограничения на минимальное число игроков. Например, волейбольная команда не может иметь меньше пяти игроков.

4.3. Кроме ограничений целостности связи ограничения, охватывающие несколько таблиц, могут представлять собой предложения, проверяющие от-

сутствие логических противоречий между данными взаимосвязанных таблиц. Например, если для каждой должности установлена определенная вилка оклада, то значение поля «Оклад» в таблице «Кадры» не должно выходить за пределы этой вилки, которая зафиксирована в таблице «Должности».

5. Особым видом ограничений являются ***алгоритмические зависимости*** между показателями. Если в БД хранятся как исходные, так и производные показатели, то необходимо поддерживать соответствие между ними. Следует обратить внимание на особенности корректировки производных показателей: обычно их корректировка либо должна быть запрещена, либо она должна быть следствием изменения исходных показателей.

6. Своеобразным видом ограничения является ***запрет на обновление***. Он может относиться как к отдельному полю, так и к записи или целой таблице. В некоторых СУБД существует запрет на корректировку ключевого поля. Но даже в тех СУБД, которые позволяют корректировать ключевые поля, лучше стараться этого избегать.

Если объекты имеют статические свойства (на ER-диаграмме отмечены буквой «С»), то для них можно задавать запрет на обновление. Так, например, если описывается объект ЛИЧНОСТЬ, то такие атрибуты, как «Дата_рождения» и «Место_рождения» являются постоянными и меняться не могут. Задание запрета на обновление для соответствующих полей в базе данных гарантирует, что сохраненная в БД информации не будет случайно или преднамеренно искажена.

Рассмотрим следующий пример ограничения на обновление записи. Пусть в базе данных по кадровому составу для каждого сотрудника хранятся сведения об их поощрениях/наградах. Эта информация хранится в таблице «Поощрения», имеющей поля: «Табельный_номер сотрудника», «Вид_поощрения», «Дата». В эту таблицу могут добавляться записи, но каждая отдельная запись изменяться не может.

В рассматриваемом примере наблюдается также ограничение связи по существованию между таблицами «Поощрения» и «Сотрудники»: «Табельный_номер» в таблице «Поощрения» должен обязательно присутствовать в таблице «Сотрудники»; при удалении записи в таблице «Сотрудники» все связанные с ней записи в таблице «Поощрения» должны быть также удалены.

Некоторые СУБД позволяют задавать при описании данных так называемое обязательное членство для включения и каскадное удаление. В этом случае целостность при корректировке будет обеспечиваться системой автоматически.

7. Ограничения целостности можно не только накладывать, но и отменять. При этом между отношениями могут существовать зависимости, и отмена одного из них может потребовать ликвидации других (ссылочных) ограничений, зависящих от первоначального. Например, если объявлено, что в таблице, содержащей сведения об организациях, поле «Наименование_организации» является уникальным и объявлена ссылочная целостность с таблицей «Поставка», в которой также имеется это поле, а потом отменяет-

ся ограничение на уникальность поля «Наименование_организации» в первой таблице, то ссылочное ограничение целостности также должно быть удалено (поскольку ссылочная целостность проверяется только в случае, если в главной таблице соответствующее поле является ключевым). Некоторые СУБД автоматически поддерживают каскадное удаление ограничений целостности, когда при удалении одного из них удаляются все зависящие от него ограничения целостности.

Другими словами, понятие целостности можно отнести не только к БД, но и к самой системе ограничений, и было бы хорошо, чтобы СУБД ее автоматически поддерживали.

8. *По моменту контроля* за соблюдением ограничения целостности различают ***одномоментные*** и ***отложенные*** ограничения целостности. *Отложенные* ограничения целостности могут не соблюдаться в процессе выполнения какой-то группы операций, но должны быть соблюдены по их завершении.

С понятием отложенного ограничения целостности тесно связано понятие транзакции - законченной совокупности действий над БД, которая переводит БД из одного целостного в логическом смысле состояния в другое целостное состояние.

Примером отложенных ограничений целостности могут служить действия при выполнении бухгалтерских проводок: в бухгалтерском учете действует принцип двойной записи; в какой-то момент, когда проведена запись по дебету счета, но еще не проведена запись по кредиту корреспондирующего счета, может временно нарушиться баланс, но по завершении операции баланс должен соблюдаться.

9. Другим признаком классификации по временному признаку является классификация ***по режиму проверки корректности БД***. Возможны два режима проверки ограничений целостности: проверка в момент корректировки и проверка существующей БД. Назовем первый из них ***оперативным режимом,*** второй - ***аудитом*** БД.

10. *По необходимости описания* ограничения целостности могут быть ***явными*** и ***неявными***. *Неявные* ограничения целостности определяются спецификой модели данных и проверяются СУБД автоматически. Так, например, в иерархических СУБД запись-ребенок не может иметь несколько родителей, аналогично в сетевых СУБД с однотипными файлами для каждого набора может быть только один владелец. Неявные ограничения целостности обычно относятся к классу синтаксических ограничений целостности, в отличие от семантических ограничений целостности, обусловленных спецификой предметной области. К этому же классу ограничений целостности можно отнести проверку на допустимое значение даты при выборе соответствующего типа данных.

11. Рассмотренные выше виды ограничений целостности относились к ***данным пользователя***. Понятие целостности может касаться и ***служебной информации***. Для реляционных СУБД это прежде всего относится к поддер-

жанию соответствия между индексными файлами и соответствующими им индексируемыми файлами баз данных.

12. Наряду с понятием целостности базы данных может быть введено понятие ***информационной целостности банка данных,*** заключающееся в обеспечении правильности взаимосвязи всех его информационных компонентов (файлов баз данных, программных файлов, описаний форм ввода-вывода, отчетов. Следует отметить, что в некоторых СУБД, в частности Access, совокупность этих элементов называется базой данных.). Например, если для файла базы данных имеется связанная с ним форма вывода и вы удалите из файла поле, вывод которого предусмотрен в этой форме, то возникнет ошибка при работе системы. Нарушения целостности могут возникнуть, если изменить тип данных, и во многих других случаях.

Некоторые СУБД имеют специальный механизм, позволяющий отслеживать согласованность различных информационных компонентов банка данных.

Для отслеживания взаимосвязи между всеми информационными компонентами БнД должны использоваться словари данных.

С обеспечением целостности БнД в целом на настоящий момент времени дело обстоит хуже, чем с контролем целостности БД в узком смысле этого понятия. Так, немногие СУБД при изменении, например, имени поля хотя бы предоставляли метаинформацию о том, какие объекты (запросы, программные модули, формы, отчеты, правила проверки ограничений на значения) имеют ссылку на измененный объект, не говоря уже о том, чтобы автоматически внести необходимые изменения.

13. Различают логическую и физическую целостность БД. ***Логическая целостность*** - состояние БД, характеризующееся отсутствием нарушений ограничений целостности, присущих логической модели данных (т.е. неявных ограничений), и явных ограничений, заданных декларативным или процедурным путем. Выше речь шла именно о логической целостности. ***Физическая целостность*** — отсутствие нарушений спецификаций схемы хранения, а также физических разрушений данных на носителе.

Работоспособность программы может нарушиться не только при изменении структуры БД, но и при перемещении тех или иных файлов, используемых ею, в другую директорию. Это также необходимо учитывать при проектировании ИС.

Задание ограничений целостности и их проверка - важная часть проектирования и функционирования банков данных. Ограничения целостности, присущие той или иной предметной области, должны быть выявлены при обследовании и зафиксированы в инфологической модели. Вопрос о необходимости проверки ограничений целостности при функционировании БнД должен решаться на основе анализа эффективности проекта, так как в некоторых случаях для ее реализации требуются значительные затраты ресурсов.

Ограничения целостности в банках данных могут задаваться либо при описании баз данных (*декларативный способ задания*), либо в программах обработки данных (*процедурный способ задания*). Первый подход более

предпочтителен, и не только потому, что при декларативном способе задания используется более высокий уровень языковых средств, но и потому, что один раз заданные ограничения будут контролироваться при выполнении всех операций над данными.

Разные СУБД обладают различным набором средств для обеспечения целостности данных. При проектировании БнД необходимо изучить, какие возможности по контролю целостности предоставляет используемая СУБД. Если СУБД автоматически не поддерживает то или иное нужное ограничение, то обеспечение его соблюдения становится заботой пользователя (проектировщика).

В СУБД семейства xBASE основная масса ограничений целостности должна была быть определена на ЯМД, так как в ЯОД практически отсутствовали средства определения ограничений целостности данных. Часть ограничений целостности можно было задавать при создании экранных форм.

В современных СУБД многие ограничения можно описать на ЯОД. Они хранятся в схеме данных и при работе с БД поддерживаются автоматически.

Для контроля целостности БД применяется также механизм триггеров. *Триггер* - это действие, которое активизируется при наступлении указанного события (вставки, удаления, обновления записи). Триггеры специфицируются в схеме базы данных.

Более широким понятием по отношению к *триггеру* является понятие *хранимая процедура.* Хранимые процедуры описывают фрагменты логики приложения, хранятся и исполняются на сервере, что позволяет улучшать характеристики производительности.

4.2. ER-модели и ограничения целостности

Некоторые ограничения целостности видны из описания предметной области в виде ER-модели.

1. Ограничение на уникальность - уникальные идентификаторы объектов являются ключами таблиц, соответствующих этим объектам. Если идентификаторов несколько, то для СУБД, поддерживающих концепцию ключа, нужно определить первичный ключ (Primary Key) и вероятные ключи (Alternate Key, Unique).

2. Между уникальным идентификатором объекта и полями, соответствующими единичным свойствам объекта, существует функциональная зависимость.

3. Если объекты связаны между собой, то соответствующие таблицы БД могут иметь ограничение целостности по связи.

Тип связи между объектами (1:1, 1:M, M:M) определяет, что будет первичным, а что - внешним ключом в этих связях. Если отношение между объектами 1:M, то очевидно, что первичным будет ключ основной таблицы (объект, который стоит со стороны единичной связи), внешним - соответствующее ему поле в таблице, отображающей объект, стоящий на стороне множественной связи.

В случае, если связь 1:1, объекты относительно связи являются равноправными, и для определения, какой из файлов БД будет играть ведущую роль, необходимо рассматривать дополнительные характеристики. Например, если класс членства с одной стороны связи необязательный, а с другой - обязательный, то идентификатор объекта, имеющего необязательный класс членства, помещается в таблицу, соответствующую объекту с обязательным классом членства, и именно он будет внешним ключом.

При наличии связи M:M в базу данных вводится дополнительная связующая таблица. Между ней и таблицами, соответствующими исходным объектам, существуют ограничения целостности по связи. Внешними ключами будут идентификаторы связанных объектов, помещенные в связующей таблице.

4. Если при наличии связи типа 1:M класс членства в связи - необязательный для объектов, стоящих со стороны множественной связи, то могут быть «пустые» значения внешнего ключа. В противном случае они должны иметь ограничение целостности «NON NULL».

5. Класс членства объектов в связи оказывает влияние не только на выбор варианта построения логической структуры, но и на задание ограничений целостности. Причем ограничение целостности будет определяться как классом членства, так и выбранным вариантом отображения этой связи в БД. Так, если объекты связаны отношением 1:M, класс принадлежности *n*-связной сущности является необязательным и для отображения связи создается третье отношение, которое содержит ключи каждой из связанных сущностей, то между связующим файлом и основными файлами будет ограничение целостности по связи, а значения каждого из полей связующего файла не должны быть пустыми. Если же дополнительная таблица не создается, то значение поля связи, включенного в таблицу, соответствующую объекту, стоящему со стороны множественной связи, либо должно совпадать с одним из значений ключевого поля связанного объекта, либо может быть пустым. Если класс членства обязательный, то поле связи не может иметь пустого значения.

6. При наличии связи 1:1 следует проверять количество элементов в связи.

7. Для статических свойств объектов можно предусматривать запрет на обновление.

8. Если свойство - условное, то соответствующий атрибут может иметь пустые значения.

4.3. Задание ограничений целостности в ERWin

При построении ER-модели в ERWin можно задавать ограничения целостности.

4.3.1. Обязательный атрибут

Для атрибута можно задавать свойство «Required» (обязательный). Для тех атрибутов, которые выбраны в качестве первичного ключа, это свойство является неактивным, поскольку свойство обязательности и так (по определению ключа) присуще элементам ключа. Свойство «Required» следует задать для атрибутов «Фамилия», «Имя», «Отчество» объекта СОТРУДНИК, «Наименование предмета полное» объекта ПРЕДМЕТ и некоторых других атрибутов. Задание этого свойства будет означать, что при вводе данных в БД недопустимо пустое значение соответствующего поля.

4.3.2. Ограничения целостности связи

При описании связи можно задать ограничения целостности связи. Для этого следует воспользоваться вкладкой **RI Actions** (рис. 4.3) в окне редактора связей **(Relationship Editor).** В этой секции для каждой связи можно задать действия, которые будут выполняться при удалении (Delete), вставке (Insert) и обновлении (Update) как порожденной (Child), так и родительской (Parent) сущности.

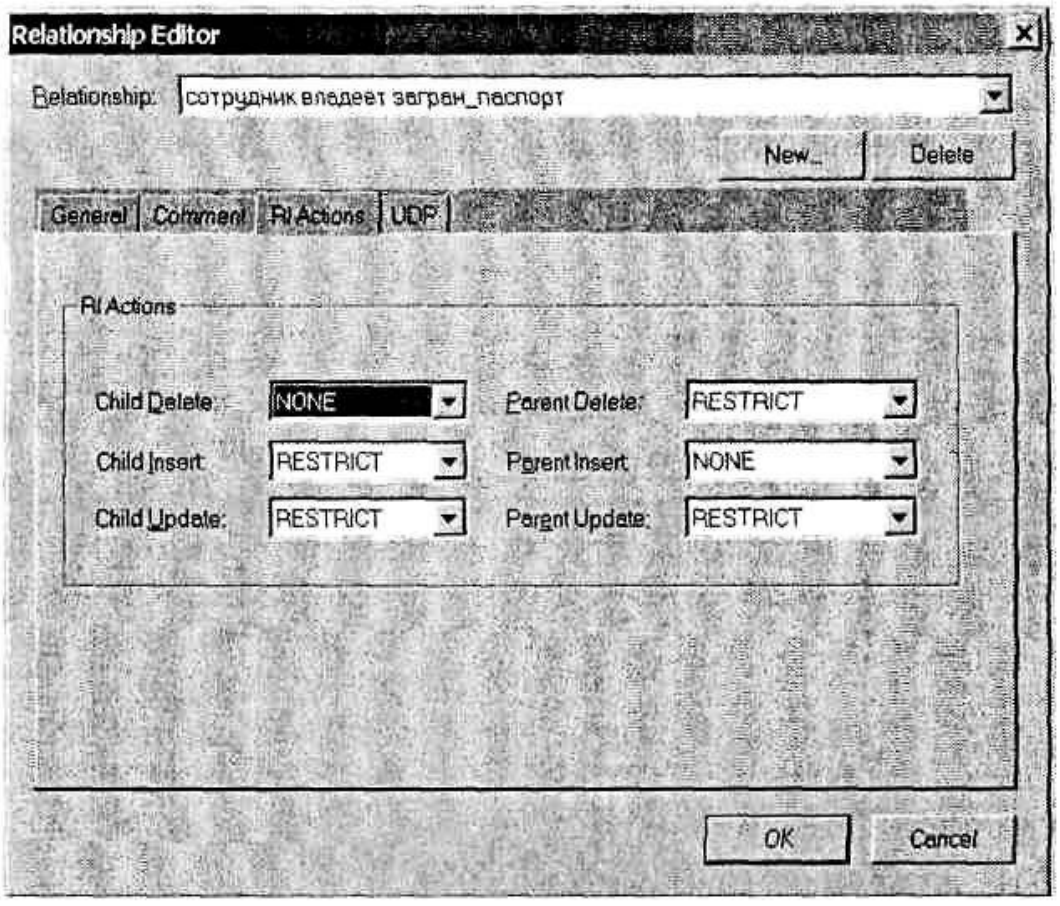


Рис. 4.3. Описание целостности связи в ERWin

Для каждой корректирующей операции можно выбрать действие, которое представлено в ниспадающих списках. Каждый список имеет четыре возможных значения: NONE (никакой), RESTRICT (ограничивать), CASCADE (каскад), SET DEFAULT (значение по умолчанию).

На рис. 4.3 показаны значения RI Actions, задаваемые по умолчанию. В рассматриваемом примере для операции Parent Delete следует выбрать действие CASCADE. Если «Код_сотрудника» может изменяться, то для операции Parent Update также следует выбрать действие CASCADE.

Так как связь «многие ко многим» в реляционной модели не поддерживается, то на уровне логической модели нет смысла (и, как следствие, нет возможности) задавать действия при корректировке сущностей, связанных таким типом связи. При необходимости можно перейти к уровню физической модели и скорректировать ограничения связи для связей, появляющихся в физической модели взамен связи «многие ко многим».

Естественно, что выбор режима действий при выполнении корректирующих операций будет зависеть от типа связи между сущностями. В табл. 4.1 приведены возможные режимы для каждого вида связи. Значения по умолчанию выделены полужирным курсивом с подчеркиванием.

Таблица 4.1

Действие	Идентифицирующая связь	Неидентифицирующая связь (Nulls Allowed)	Неидентифицирующая связь (No Nulls)	Категориальная связь
Child Delete	***NONE,*** RESTRICT, CASCADE	***NONE,*** RESTRICT, CASCADE, SET DEFAULT, SET Null	***NONE,*** RESTRICT, CASCADE, SET DEFAULT	***NONE***, RESTRICT, CASCADE
Child Insert	NONE, ***RESTRICT***, CASCADE	NONE, RESTRICT, CASCADE, SET DEFAULT, ***SET Null***	NONE, ***RESTRICT,*** CASCADE, SET DEFAULT	NONE, ***RESTRICT,*** CASCADE
Child Update	NONE, ***RESTRICT,*** CASCADE	NONE, RESTRICT, CASCADE, SET DEFAULT, ***SET Null***	NONE, ***RESTRICT,*** CASCADE, SET DEFAULT	NONE, ***RESTRICT,*** CASCADE
Parent Delete	NONE, ***RESTRICT,*** CASCADE,	NONE, RESTRICT, CASCADE, SET DEFAULT, ***SET Null***	NONE, ***RESTRICT*** CASCADE, SET DEFAULT	NONE, RESTRICT, ***CASCADE***

Продолжение таблицы 4.1

Parent Insert	***NONE,*** RESTRICT, CASCADE	***NONE,*** RESTRICT, CASCADE, SET DEFAULT, SET Null	***NONE,*** RESTRICT, CASCADE, SET DEFAULT	***NONE,*** RESTRICT, CASCADE
Parent Update	NONE, ***RESTRICT,*** CASCADE	NONE, RESTRICT, CASCADE, SET DEFAULT, ***SET Null***	NONE, ***RESTRICT,*** CASCADE, SET DEFAULT	NONE, RESTRICT, ***CASCADE***

Значения ограничений означают следующее:

- NONE - действие не оказывает влияния на связанные записи;
- RESTRICT - действие запрещено (при определенных условиях);
- CASCADE - действие вызывает изменения в связанных записях;
- SET DEFAULT - устанавливается значение по умолчанию для поля связи;
- SET Null - устанавливается по умолчанию значение Null для поля связи.

Обычно удаление зависимой записи (Child Delete) при любом типе связи не требует дополнительных проверок на допустимость корректировки и не приводит к каким-либо изменениям в связанных записях. Поэтому режимом по умолчанию для всех типов связи является NONE. Дополнительная проверка может потребоваться, если в предметной области невозможно существование главного объекта без связанных с ним подчиненных объектов. Например, не может быть ОТДЕЛА без СОТРУДНИКОВ. В этом случае либо не должна быть допущена попытка удаления последнего сотрудника из отдела, либо при этом должен быть удален отдел.

Как правило, вставка зависимой записи при наличии идентифицирующей связи не может быть проведена, если отсутствует соответствующая ей запись в главной таблице (ситуация, когда в главной таблице нет записи со значением ключа, равным введенному значению поля связи), т.е. используется режим RESTRICT. Можно представить себе такую организацию ведения таблиц базы данных, когда в основную таблицу при вводе зависимой записи автоматически вводится новая запись с ключом, соответствующим значению поля связи зависимой записи. Но такой возможностью не нужно злоупотреблять, так как отсутствующее значение может быть вызвано ошибкой при вводе данных, а не отсутствием информации в базе данных. Режим NONE при вставке зависимой записи при наличии идентифицирующей связи использовать не следует. Если в предметной области возникает необходимость вставить зависимый объект, не связанный с основным объек-

том (т.е. класс членства объекта в связи — необязательный), то для связи таких объектов следует выбрать неидентифицирующую связь.

Так называемая категориальная связь является особой связью: с одной стороны, она является идентифицирующей связью, а с другой - связывает не два разных объекта, а информацию об одном и том же объекте. Для обобщенного объекта хорошо бы иметь специальный инструмент, который позволял бы рассматривать его как единое целое. В ERWin можно задавать ограничения целостности только для каждой отдельной связи, соединяющей родовой объект с каждым из видовых объектов.

4.3.3. Триггер ссылочной целостности

Для обеспечения ссылочной целостности может быть создан особый вид триггера - триггер ссылочной целостности. По умолчанию ERWin генерирует триггеры, обеспечивающие контроль ссылочной целостности для каждой связи, определенной в ER-модели.

При генерации триггеров ERWin использует механизм шаблонов - специальных скриптов, использующих макрокоманды.

Шаблоны триггеров ссылочной целостности, используемые ERWin, можно изменять, причем можно переопределить как триггеры для конкретной связи, так и шаблоны для всей модели в целом.

Для редактирования триггера следует (находясь в физической модели) щелкнуть правой кнопкой мыши по таблице и выбрать во всплывающем меню пункт [название целевой СУБД] Trigger (рис. 4.4).



Рис. 4.4. Позиция меню для редактирования триггера

Поскольку не все СУБД поддерживают механизм триггеров, то соответствующая позиция меню активна не всегда: она не активна для настольных СУБД и активна для корпоративных систем.

На это следует обратить внимание

- Обеспечение целостности базы данных является важнейшей задачей при проектировании и эксплуатации БД.
- При проектировании БД необходимо выявить все ограничения целостности, которые обусловлены как особенностями предметной области,

так и спецификой используемого программного и технического обеспечения.

- Из общего списка ограничений целостности необходимо выявить те, которые будут контролироваться в данной ИС, и определить механизм их реализации.

Контрольные вопросы

1. Что такое «ограничения целостности»?
2. В чем важность задания ограничений целостности?
3. Какие виды ограничений целостности вы знаете? Приведите примеры ограничений целостности каждого вида.
4. Какие способы задания ограничений целостности вы знаете? В чем преимущества и недостатки каждого из них?
5. В чем суть применения триггеров для контроля целостности данных?
6. В какой момент происходит проверка соблюдения ограничения целостности, относящегося к полю? ограничения, задающего отношение между значениями разных полей одной записи? отложенных ограничений целостности?
7. Что такое ограничение целостности связи?
8. Если задано ограничение целостности связи, но не задано каскадное удаление связанных записей, повлияет ли заданное ограничение целостности на процесс удаления записи из основного файла?
9. Если задано ограничение целостности связи, может ли значение внешнего ключа быть пустым?
10. В чем суть ограничения целостности по существованию? В чем его отличие от ограничения целостности связи?
11. Какие виды диапазонов вы знаете? В чем особенности их задания?
12. Что такое «домен»? Как можно реализовывать ограничения целостности на «домен»?
13. Что такое «транзакция»? С каким видом ограничений целостности обычно связаны транзакции?
14. Какие ограничения связи непосредственно отображены в ER-модели?
15. Какие ограничения целостности и каким образом могут быть заданы в ERWin?

Глава 5 СОЗДАНИЕ И ВЕДЕНИЕ БАЗ ДАННЫХ

5.1. Описание структуры баз данных.

Общие сведения

Спроектированная структура базы данных должна быть описана на языке описания данных или с использованием соответствующих операторов интегрированного языка. Язык описания данных может быть различным: аналитическим, табличным, смешанным. Описания данных могут храниться отдельно от данных или вместе с ними. В СУБД предшествующего поколения (к которым относятся иерархические и сетевые системы) обычно использовалось раздельное хранение данных и их описаний. В большинстве современных настольных реляционных СУБД описания БД хранятся вместе с данными.

В связи с тем, что наибольшее развитие в настоящее время имеют реляционные СУБД, рассмотрим вопросы создания БД на их примере.

Для описания логической структуры данных в настольных реляционных СУБД обычно используется простой табличный язык описания данных, физические параметры практически не указываются. Загрузка данных в базу данных заключается либо во вводе данных с клавиатуры, либо в переносе их из других файлов. Наблюдается упрощение технологического процесса, связанного с созданием баз данных.

Наиболее часто используемым способом создания файла/таблицы базы данных является выбор из соответствующего меню позиции «создать новый файл базы данных/таблицу». Возможно и использование соответствующего оператора языка описания данных. В тех системах, в которых наряду с понятием «файл базы данных/таблица» поддерживается и понятие «база данных», нужно сначала определить базу данных, а затем - таблицы, входящие в нее.

Как указывалось выше, обычно описание структуры таблиц само представляется в табличной форме. Каждая строка этой таблицы описания соответствует одному полю таблицы. Каждое поле должно иметь уникальное имя в пределах таблицы. Максимальная длина этого имени и символы, допустимые при его задании, зависят от используемой операционной среды и СУБД.

Каждое поле имеет определенный тип (Field Type). Современные СУБД обычно позволяют при описании поля просмотреть список допустимых типов полей и выбрать нужный. Кроме того, во многих СУБД задать нужный тип поля можно, указав первую букву названия выбираемого типа.

Допустимые типы полей различаются от системы к системе. Наблюдается тенденция к расширению списка поддерживаемых типов данных. Многие современные СУБД наряду с традиционными типами полей (Character - символьное, Numeric - числовое, Floating - с плавающей запятой, Date - дата, Logical - логическое, Memo - поле памяти) поддерживают и такие экзотические до недавнего времени типы полей, как двоичные, рисунки и др.

Наиболее развитые системы (например, Oracle7) позволяют в единой базе данных интегрирование хранить не только структурированные данные, но и неструктурированные, такие, как поточная аудио- и видеоинформация, пространственные многомерные данные.

Для массовых применений наиболее часто используются типы Character, Numeric, Logical и Date. Тип поля Floating в основном применяется для математических расчетов, тип Memo - для хранения данных, длина которых резко отличается от записи к записи, а также при хранении длинных полей. Этот тип поля называют также полем примечания, а иногда и просто текстовым.

После указания типа поля обычно указывается его длина. Для некоторых типов полей длина предопределена и устанавливается системой автоматически. Так, например, для полей типа даты автоматически устанавливается длина, равная 8 символам, для логического типа - 1 символу. Для других типов полей проектировщик при описании файла должен задать длину поля. В некоторых случаях эта величина может быть составной. Так, например, если это поле типа Numeric, то указывается общая длина поля, включая десятичную точку и знаки после нее, а также задается число знаков после десятичной точки.

Для некоторых типов полей, например для поля типа Мемо, в качестве длины иногда указывается длина не собственно этого поля, а длина указателя, который будет храниться в записи реляционной БД и указывать на соответствующий блок в отдельном текстовом файле. Последний автоматически создается системой для хранения Мемо-полей (как это реализовано в dBase) или не указывается вообще (как в Access).

Многие реляционные системы позволяют при описании структуры файла БД указать и признаки индексирования по отдельным полям или по совокупности полей (составной индекс). Соответствующие колонки таблицы описания структуры файла БД/таблицы в разных системах называются поразному (Index, Tag). В некоторых СУБД (например, Access) индексирование задается как свойство поля.

Индексация представляет собой способ логической упорядоченности файлов. Если при создании файла для какого-то поля в графе индексирования установлено значение «да», то для этого поля создается специальный индексный файл, который содержит значения этого поля и номер записи в индексируемом файле, содержащей такое значение поля. Индексный файл является упорядоченным (обычно организован как двоичное дерево сортировки). Многие системы позволяют также при описании файла указывать и вид упорядочения (по возрастанию, по убыванию).

Если значение «да» в графе признака индексации установлено для нескольких полей, то для каждого из них создается аналогичный файл (в зависимости от реализации хранения индексов в конкретных СУБД все эти индексные файлы могут объединяться в один мультииндексный файл или храниться как самостоятельные отдельные файлы).

Кроме индексации по одному полю можно проводить индексацию по совокупности полей (составной индекс) или по более сложному выражению.

Нужно быть внимательными при задании индексации по сложному выражению, в котором участвуют поля разных типов, поскольку некоторые СУБД самостоятельно проводят необходимые преобразования типов, а некоторые - нет (при этом в лучшем случае может быть выдано предупреждающее сообщение, а в худшем - можно получить не тот результат, на который рассчитывали, и не сразу догадаться об этом).

После того как описано одно поле, переходят к описанию следующего, и так до тех пор, пока не будут определены все поля данного файла БД.

Реляционные СУБД должны поддерживать концепцию ключа. Некоторые СУБД требуют, чтобы при описании таблицы обязательно был указан ее ключ, другие - предоставляют такую возможность, но не требуют, чтобы в каждой таблице ключ был обязательно задан, третьи - вообще не предоставляют возможности идентифицировать ключ. Те системы, которые поддерживают концепцию ключа, обычно не только обеспечивают проверку на его уникальность, но и автоматически проводят индексацию по ключевому полю. (В Access, например, таблица, для которой определен ключ, так и называется индексированной.) Некоторые СУБД, например та же Access, при сохранении новой таблицы, для которой не задан ключ, предлагают определить ключ автоматически и в случае положительного ответа формируют дополнительное поле-счетчик с именем «Код», которое будет содержать уникальное значение для каждой записи. Такое решение можно принять, когда запись таблицы не содержала естественного ключа или когда естественный ключ слишком длинный.

В некоторых системах при описании файлов БД можно задать и ограничения целостности. Тенденции развития СУБД таковы, что даже настольные системы включают все более развитые возможности контроля целостности.

Кроме рассмотренных выше свойств полей СУБД позволяют определять и другие свойства, такие, как подпись поля (название поля, которое будет использоваться при выводе информации), формат поля и др.

Многие СУБД предоставляют возможность использовать и иные способы создания таблиц, не предполагающие непосредственного описания каждого поля проектировщиком.

Так, при создании новой таблицы можно воспользоваться образцами, которые обычно включаются в состав СУБД. Но это не освобождает от понимания основ проектирования БД, так как следует внимательно оценить, насколько предлагаемое в качестве образца решение соответствует вашим потребностям, и при необходимости изменить предлагаемую структуру БД.

Кроме того, имеются возможности полного или выборочного копирования структуры имеющегося файла при создании нового файла.

Имеются и другие способы создания таблиц БД. Так, многие СУБД позволяют проводить импорт файлов/таблиц, созданных в других приложениях, в создаваемую БД. При этом описание включенной в БД таким образом таблицы будет сформировано автоматически.

При завершении описания Структуры файла БД необходимо дать имя этому файлу (таблице).

Следует обратить внимание на то, что база данных представляет собой не просто совокупность отдельных файлов, а единое взаимосвязанное хранилище данных. Поэтому описание БД не ограничивается описанием отдельных ее файлов (таблиц). В СУБД, поддерживающих нереляционные структурированные модели данных (иерархические, сетевые), описание связей между информационными единицами является неотъемлемой частью описания БД. В реляционных СУБД (в силу специфики модели БД) описание связей в схеме БД не является обязательным. Однако многие СУБД позволяют описывать связи при описании БД; это прежде всего служит цели задания ограничений целостности.

5.2. Создание БД в Microsoft Access

Рассматривая работу в среде MS Access, еще раз обратим внимание на использующуюся в системе терминологию. Базой данных в MS Access называется совокупность таблиц, форм, отчетов, запросов, модулей, макросов. Вся эта совокупность запоминается в одном файле базы данных, имеющем расширение (.mdb).

В контексте данного раздела под «созданием баз данных» будем понимать только создание таблиц и связей между ними. Другие компоненты, включаемые в понятие базы данных в MS Access, здесь рассматриваться не будут.

Если вы создаете новую базу данных, то надо выбрать позиции меню **Файл/Создать** базу данных и в появившемся окне Создание выбрать позицию «База данных» (рис. 5.1).

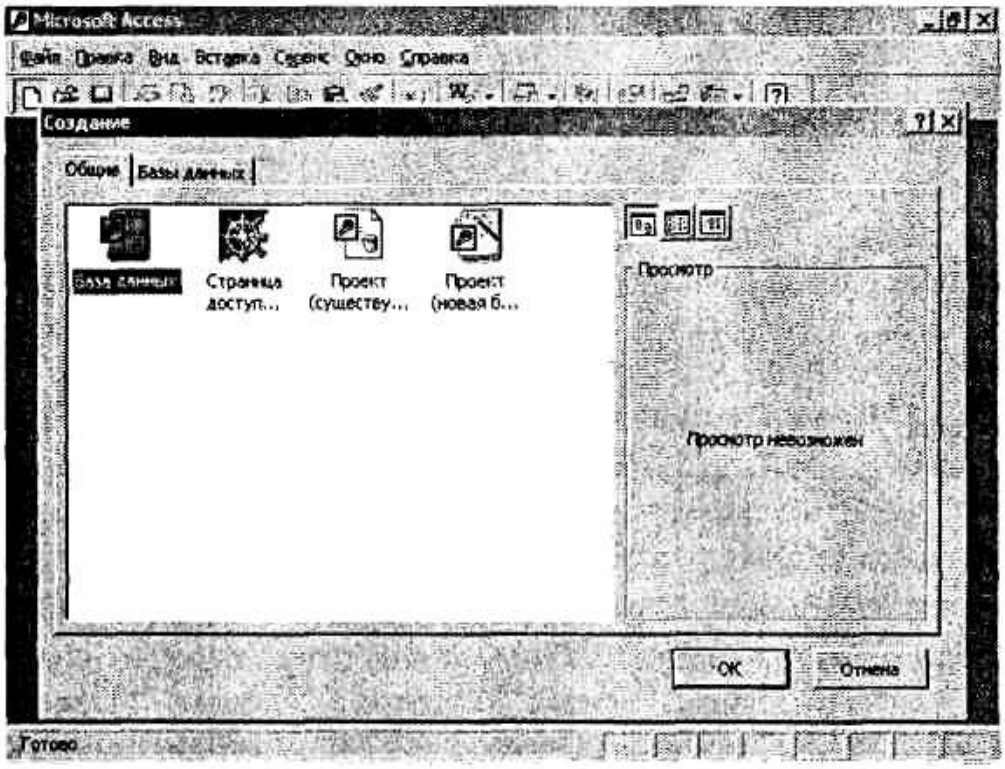


Рис. 5.1. Создание новой базы данных (экран 1)

В появившемся окне **Файл новой базы данных** (рис. 5.2) нужно задать имя файла БД и щелкнуть по кнопке **Создать.**

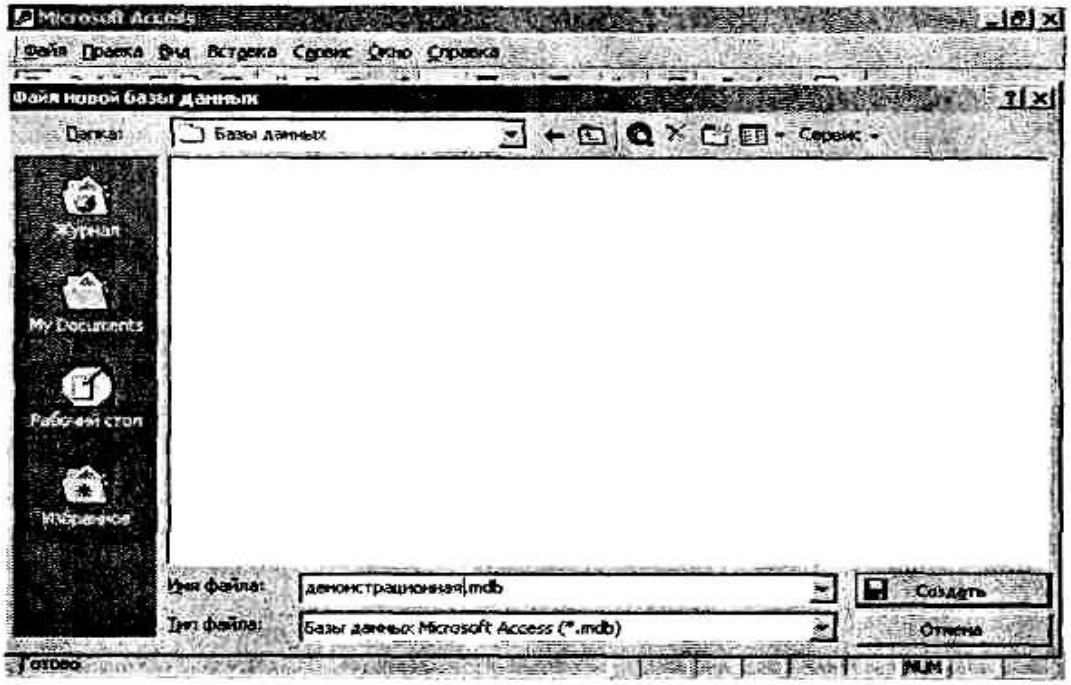


Рис. 5.2. Создание новой базы данных
(экран 2 - задание имени базы данных)

После выполнения этих шагов появится экран [название]**: база данных** (рис. **5.3).**

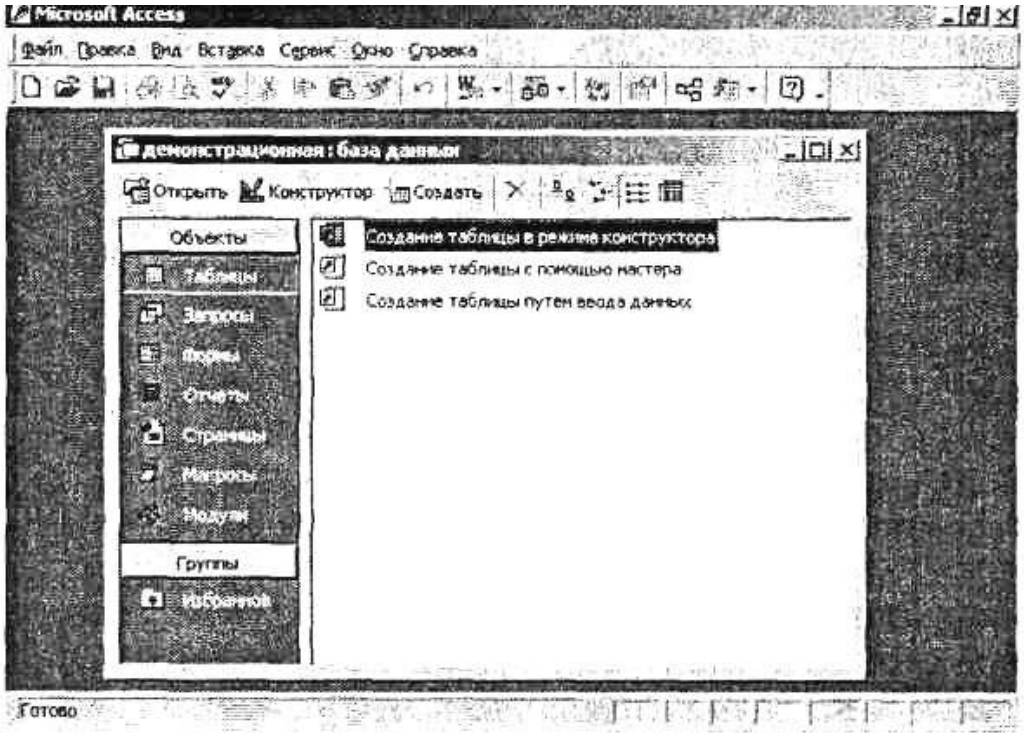


Рис. 5.3. Начальный вид окна базы данных

5.2.1. Создание новой таблицы путем описания ее структуры

Создание базы данных начинается с создания таблиц, в которых и хранится информация о предметной области. База данных обычно включает несколько взаимосвязанных таблиц. Для создания новой таблицы в окне **Базы данных** нужно выбрать вкладку **Таблица** и щелкнуть по кнопке **Создать** - появится окно **Новая таблица** (рис. 5.4).

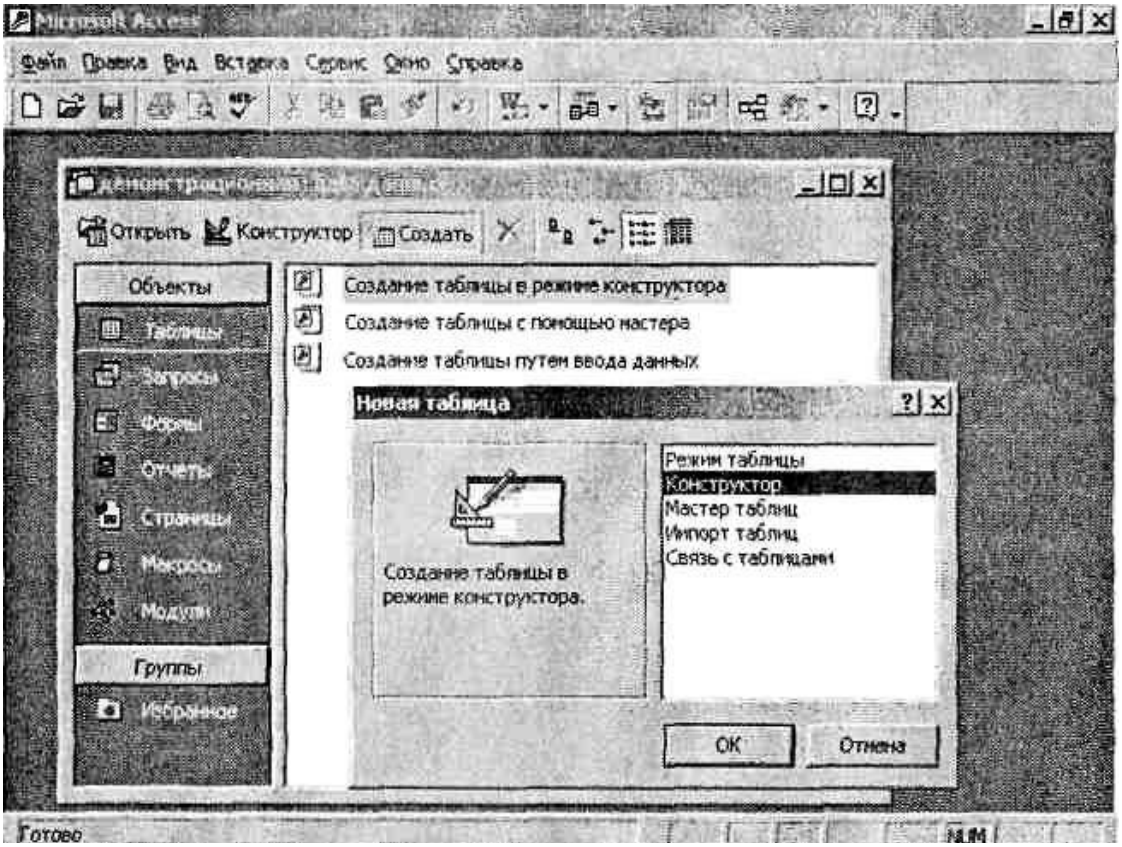


Рис. 5.4. Выбор способа создания таблицы

Создать таблицу можно в разных режимах: режиме таблицы, конструктора, мастера таблиц, импорта таблиц и связи с таблицами. Начнем рассмотрение возможностей создания таблиц с режима конструктора как наиболее часто используемого. Для этого в появившемся окне выберите режим создания нового объекта - **Конструктор**. После чего появится окно для описания структуры таблицы и других ее характеристик (рис. 5.5).

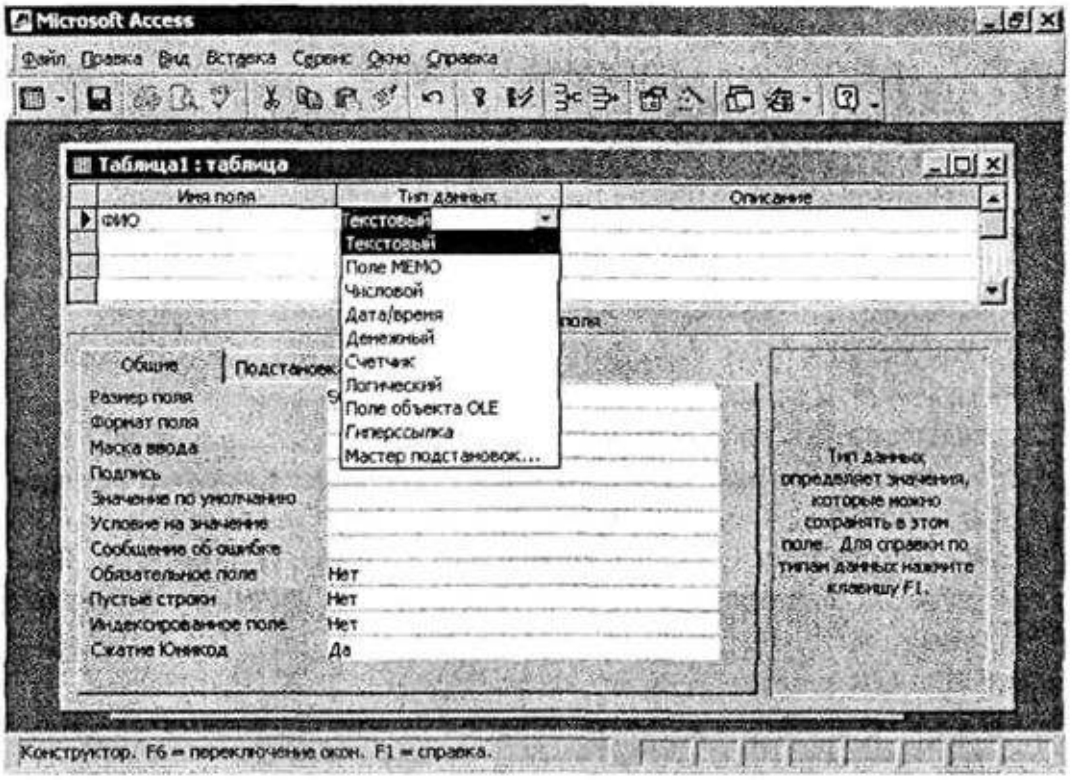


Рис. 5.5. Вид экрана при описании таблицы

Описание полей таблицы

В табличной форме необходимо последовательно описать все поля создаваемой таблицы. Сначала задается имя поля. Access допускает задание длинных имен с пробелами на русском языке.

В Microsoft Access действуют следующие ограничения на имена полей:

- имя должно содержать не более 64 символов;
- имя может включать любую комбинацию букв, цифр, пробелов и специальных символов за исключением точки (.), восклицательного знака (!), надстрочного символа (') и прямых скобок ([ ]);
- имя не должно начинаться с символа пробела;
- имя не должно включать управляющие символы (с кодами ASCII от 0 до 31).

Хотя пробелы внутри имен полей и являются допустимыми, они могут при некоторых обстоятельствах вызывать конфликты при работе с другими системами. Поэтому их не рекомендуется использовать. Вообще к заданию длинных имен на русском языке нужно относиться с осторожностью, особенно если есть вероятность, что создаваемое приложение будет в дальнейшем использоваться в распределенных гетерогенных системах.

Имена полей не должны совпадать с зарезервированными словами. Например, не следует давать полю имя Count, Name и т.п.

Имя поля должно быть уникальным в пределах таблицы. И хотя система не запрещает использование одинаковых имен полей в разных таблицах, из-

бегайте использования одинаковых имен для обозначения разных по смыслу атрибутов.

Имя должно быть понятно не только в контексте данной конкретной таблицы. Так, например, если в таблице «Сотрудник» есть поле «Код» и такое же поле есть в таблице «Кафедра», то в первом случае это будет код сотрудника, а во втором - код кафедры. Многие системы (и Access в том числе) при соответствующей настройке автоматически связывают таблицы по полям, которые имеют одинаковые имя, тип и длину. Если имена даны непродуманно, то могут возникнуть неправильные связи либо процесс задания связей будет несколько сложнее, чем при правильном задании имен.

После задания имени нужно выбрать тип поля. Если щелкнуть мышью по свободной ячейке графы «Тип поля», то высветится список допустимых типов полей (*см.* рис. 5.5), из которого и следует выбрать подходящий для описываемого поля тип. Имя и тип поля должны задаваться обязательно.

Графа «Описание» может не заполняться. Эта графа используется в целях документирования проекта.

Допустимые типы полей в Access 2000 и их краткая характеристика приведены в табл. 5.1.

Таблица 5.1

Тип данных	Содержимое поля	Размер
Текстовый	Текст или числа, не требующие проведения расчетов (номера телефонов, коды и т. п.)	Максимальное число символов - 255
Поле Мемо	Длинный текст или комбинация текста и чисел	До 65 535 символов
Числовой	Числовые данные, используемые для проведения расчетов	1,2,4 или 8 байт
Дата/время	Даты и время, относящиеся к годам с 100 по 9999 включительно	8 байт
Денежный	Специальный формат для представления числовых данных. Точность до 15 знаков в целой и до 4 знаков в дробной части	8 байт
Счетчик	Уникальные последовательно возрастающие (на 1) или случайные числа, автоматически вводящиеся при добавлении каждой новой записи в таблицу	4 байта
Логический	Поля, которые могут содержать одно из двух возможных значений (True/False, Да/Нет)	1 бит

Продолжение таблицы 5.1

Поле объекта OLE	Объект, связанный или внедренный в таблицу Microsoft Access	До 1 Гбайта (ограничивается объемом диска)
Гиперссылка	Строка, состоящая из букв и цифр и представляющая адрес гиперссылки	Может содержать до 3х2048 символов

В списке допустимых типов полей имеется строка «Мастер подстановок». При его использовании можно создать поле, содержание которого формируется путем выбора значений из списка, содержащего набор постоянных значений или значений из другой таблицы/запроса. Если источником для подстановки выбран столбец другой таблицы, то тип и длина поля, созданного таким способом, будут определяться типом и длиной элементов, служащих источником для подстановки значений.

Выбор типа поля является важным шагом, при проектировании БД. Принятое решение оказывает влияние на выполняемый при вводе контроль правильности данных, на допустимые операции над данными и особенности их выполнения, требуемый объем памяти, скорость выполнения операций, совместимость разных частей БД при работе в гетерогенной среде.

Предположим, что создается таблица, содержащая сведения о профессорско-преподавательском составе. Состав и тип полей создаваемой таблицы представлены на рис. 5.6.

Введем описания всех полей, кроме поля «Должность», выбирая указанные на рис. 5.6 типы полей.

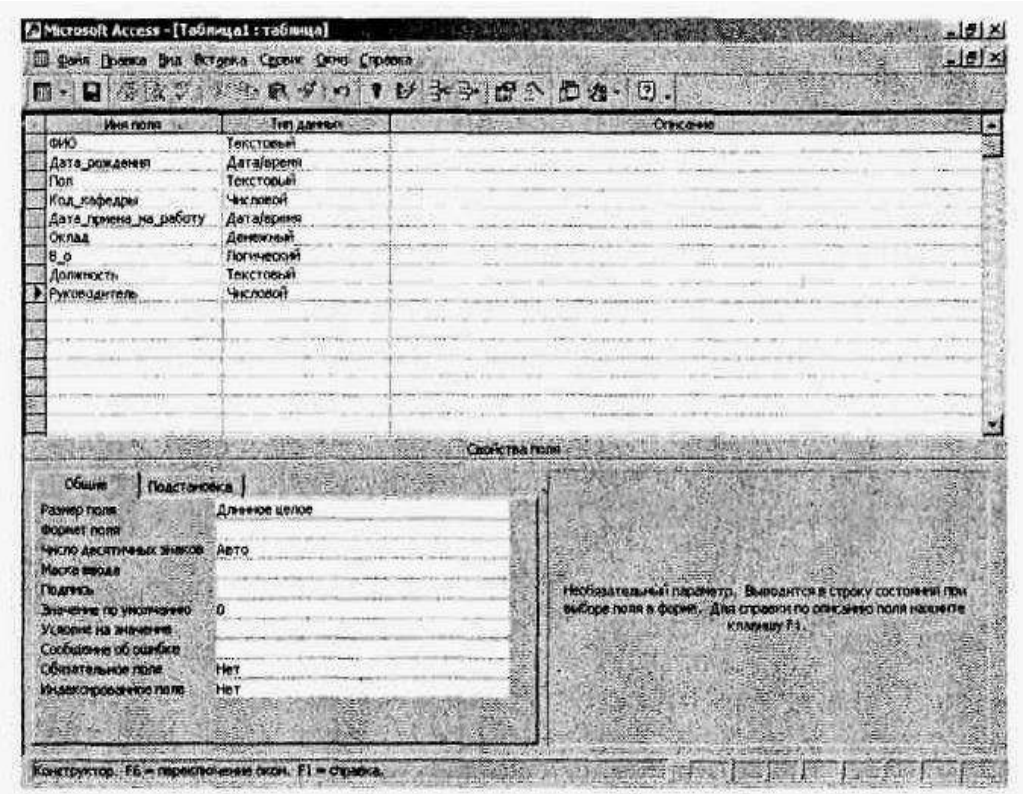


Рис. 5.6. Состав полей таблицы «Сотрудник»

Поле «Должность» создадим несколько иначе. Для выбранной категории сотрудников имеется всего четыре возможные должности: ассистент, старший преподаватель, доцент и профессор. Хорошо было бы заменить ввод этих значений выбором их из списка. В ранних версиях Access задавать домен (либо путем прямого ввода списка значений, либо путем связи с файлом подстановки) можно было только при создании запроса или экранной формы. В последних версиях стало возможным задать это и при описании таблицы.

Создадим поле «Должность», используя «Мастер подстановок». Для этого можно либо при выборе типа указать «Мастер подстановок» (*см.* последнюю строку в ниспадающем списке типов полей на рис. 5.5), либо выбрать позицию «Поле подстановки» в меню **Вставка**.

Последовательность шагов при создании поля подстановки отражена на рис 5.7-5.9. Имя поля сначала задавать не надо, оно будет задано позже в процессе создания поля с помощью мастера. Следует сразу перейти к столбцу «Тип данных» и выбрать в списке строку «Мастер подстановок».

Так как список, создаваемый в рассматриваемом случае, короткий и стабильный, то создадим столбец подстановки с фиксированным набором значений (рис. 5.7). В появившемся далее окне введем требуемые значения (рис. 5.8).

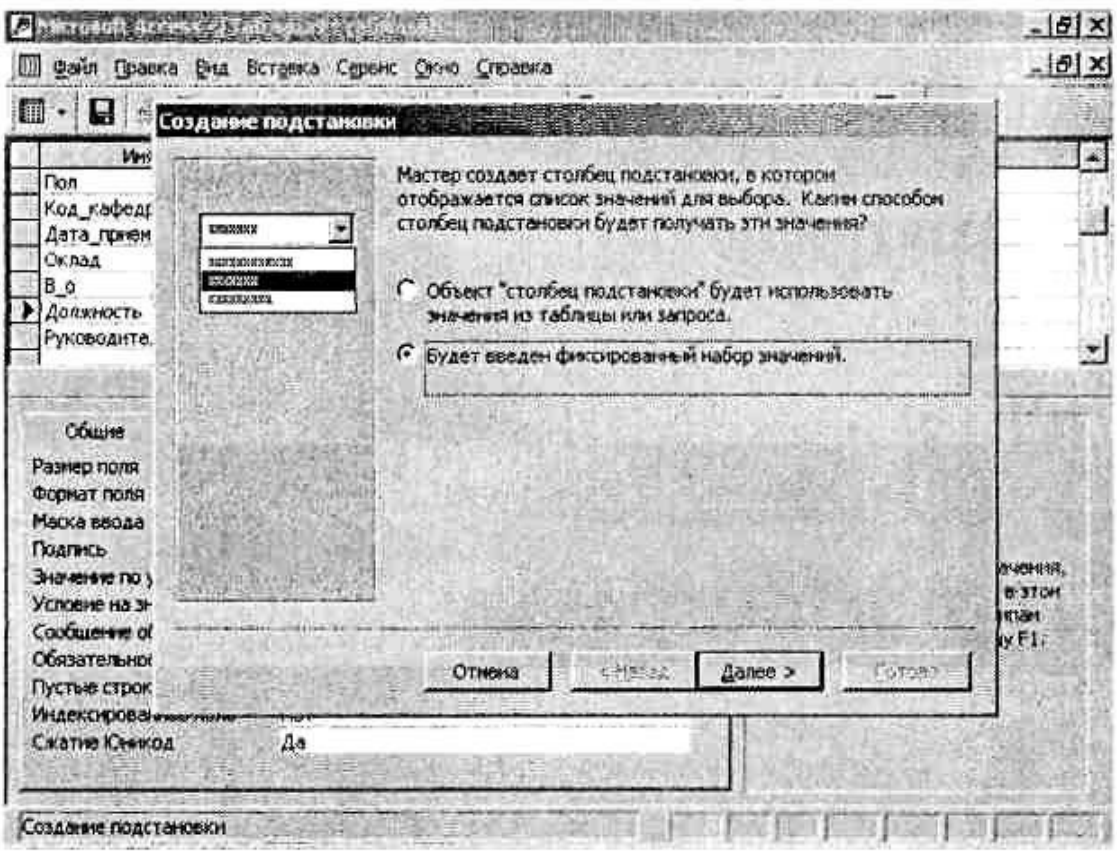


Рис. 5.7. Создание столбца подстановки. Начальный экран

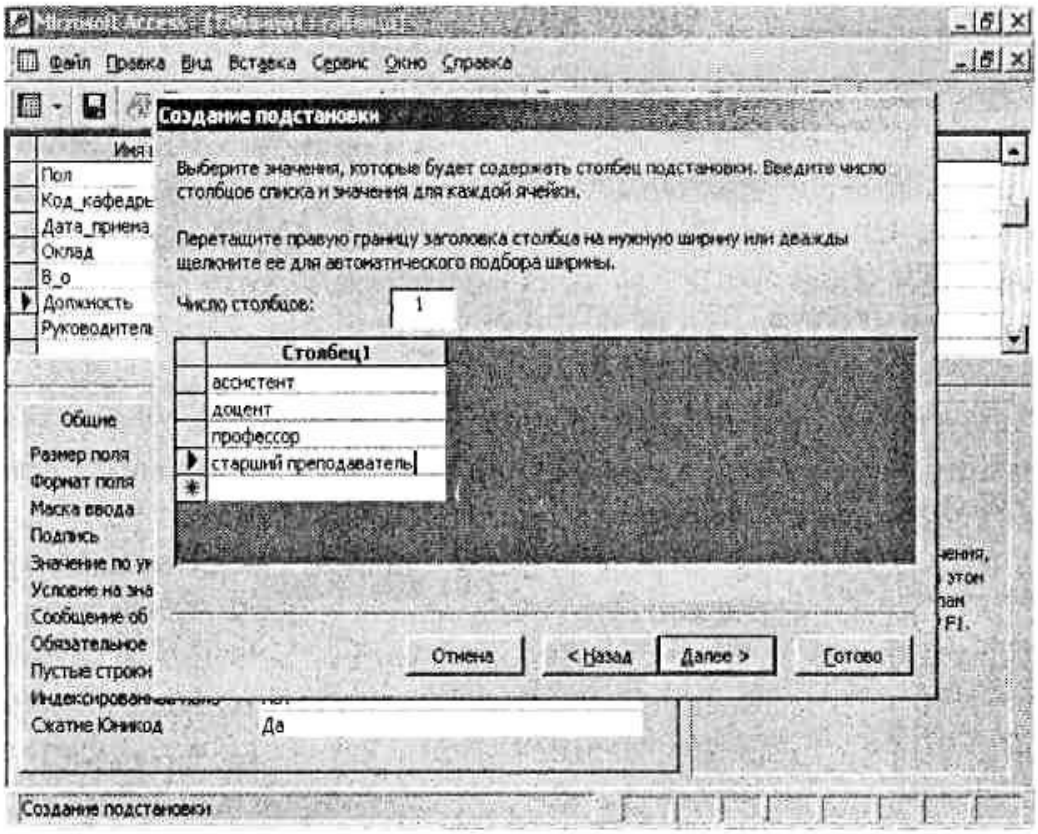


Рис. 5.8. Создание столбца подстановки.
Столбец с введенным списком значений

Далее зададим имя этого поля (рис. 5.9).

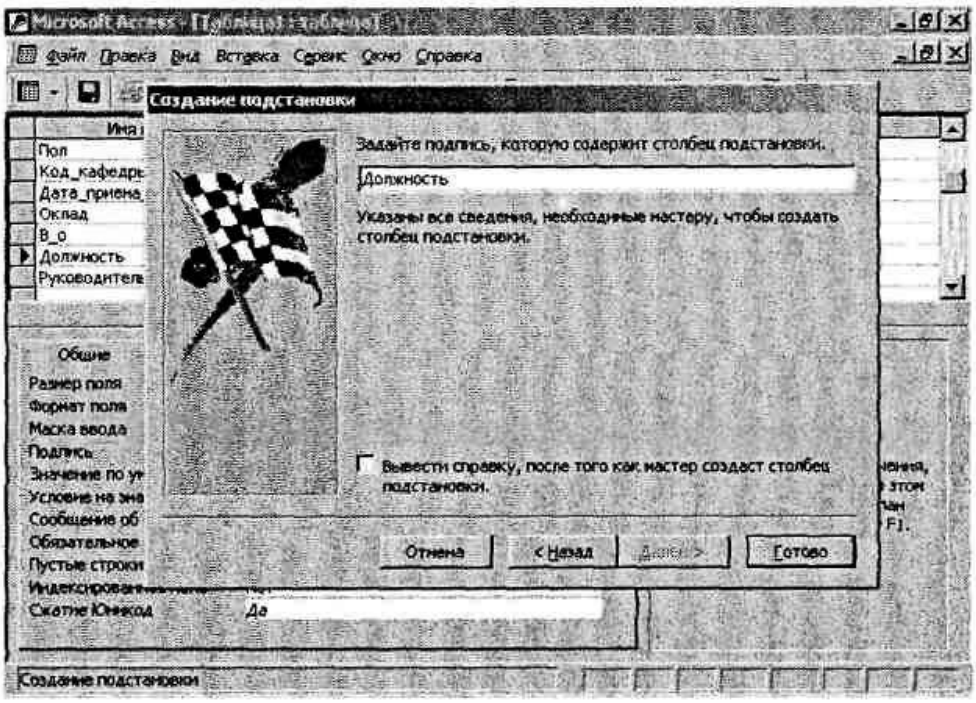


Рис. 5.9. Создание столбца подстановки. Задание имени столбца

После создания поля таким способом его тип будет «текстовый» и длина - 50. После создания поля с использованием мастера подстановок с фиксированным набором значений его длину можно скорректировать.

При вводе данных в таблицу значения полей подстановки можно не вводить с клавиатуры, а выбирать из заданного списка. Чтобы нельзя было ввести значения, отсутствующие в списке, надо в свойствах поля на вкладке **Подстановка** в позиции «Ограничиться списком» задать значение **Да**. В этом случае использование поля подстановки обеспечит не только более эффективный ввод данных, но и более жесткий контроль целостности базы данных.

Если число значений поля подстановки достаточно велико и сами значения могут меняться со временем, то следует использовать вторую альтернативу - значения из другой таблицы/запроса (рис. 5.10). Эта возможность реализуется в нашем примере для поля «Код_кафедры», значения которого будут выбраны из таблицы «Кафедра».

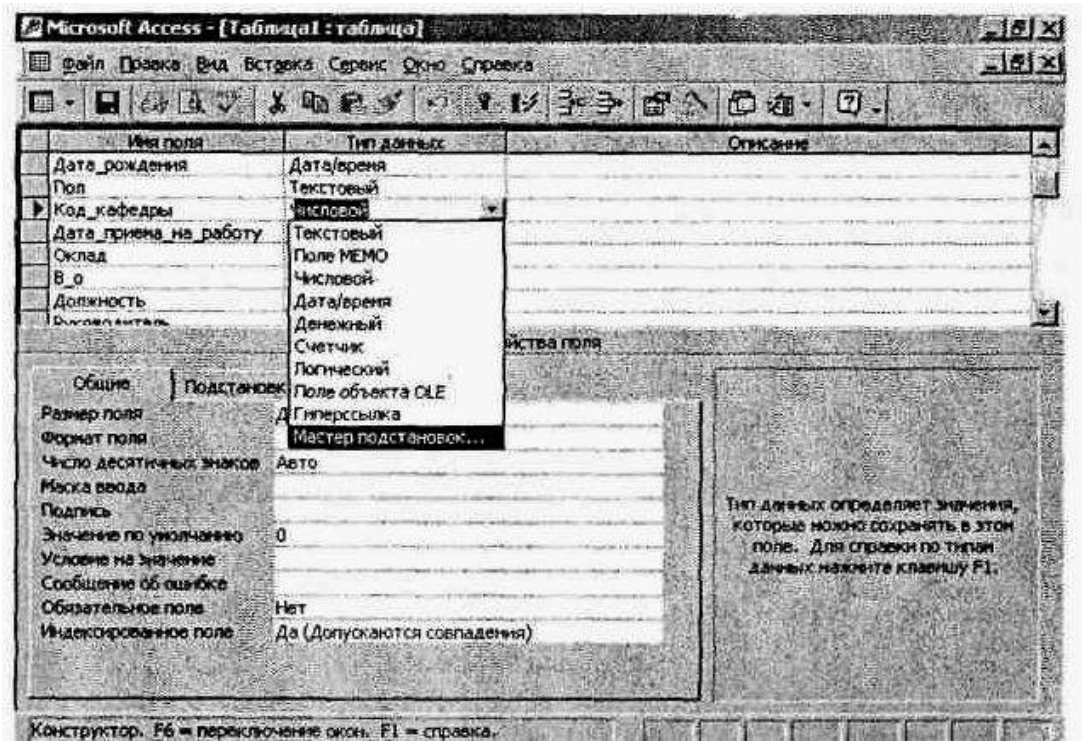


Рис. 5.10. Создание поля подстановки (шаг 1)

Для создания поля подстановки, источником для которого служит другая таблица, необходимо сначала создать основную таблицу (в паре «Кафедра»-«Сотрудник» основной будет таблица «Кафедра»), связать таблицы и только потом создавать поле подстановки (в нашем случае это поле «Код_кафедры» в таблице «Сотрудник»). Если, как в нашем случае, основная таблица еще не создана, то нужно закончить описание таблицы «Сотрудник», сделать указанные выше шаги (создание таблицы «Кафедра» и связывание ее с таблицей «Сотрудник», а потом можно скорректировать описание поля «Код_Кафедры» в таблице «Сотрудник»), выбрав в столбце «Тип данных»

позицию «Мастер подстановок» (рис. 5.10) и далее выполнить шаги, которые описаны ниже*.

В окне **Создание подстановки** выбираем альтернативу «Столбец подстановки использует значения из таблицы или запроса» (рис. 5.11).

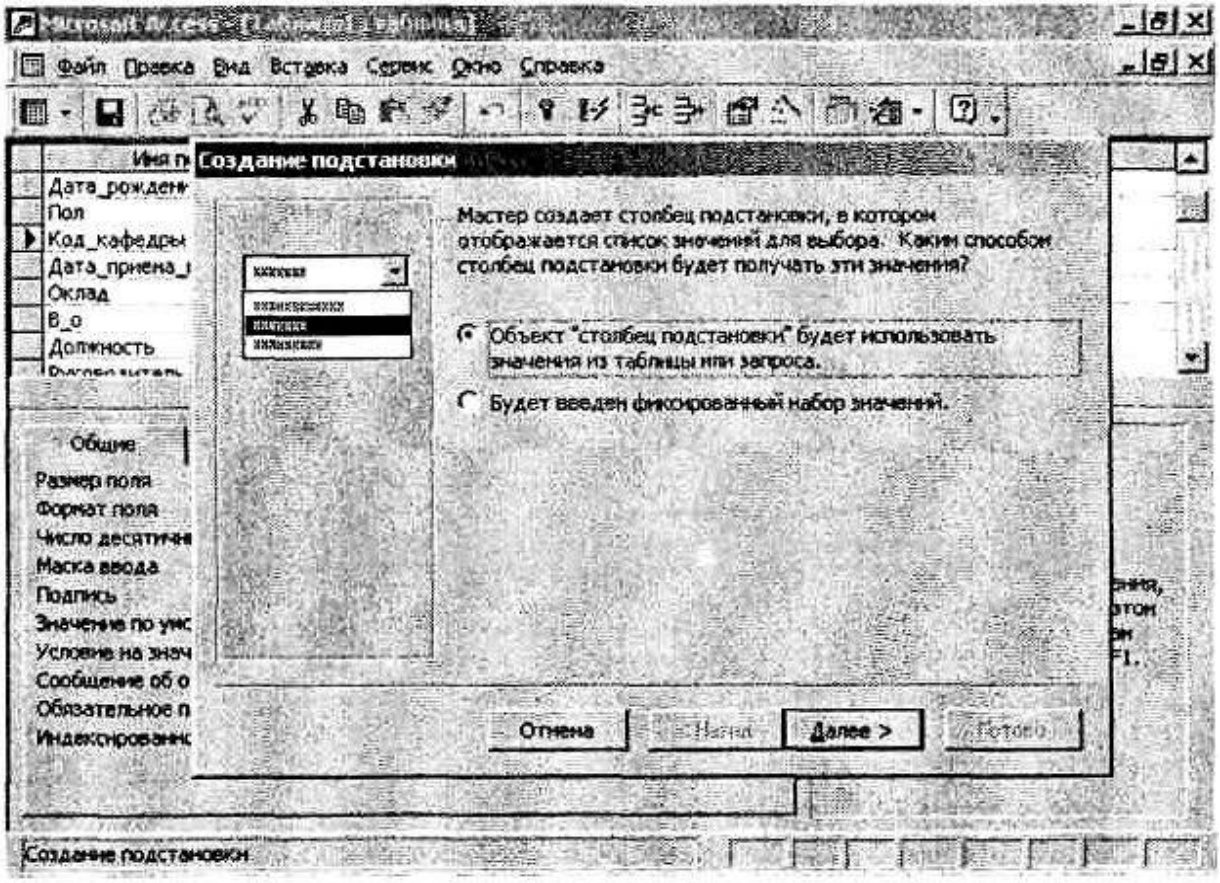


Рис. 5.11. Создание поля подстановки (шаг 2)

После этого нужно выбрать таблицу/запрос, которая будет являться источником данных для описываемого поля (рис. 5.12). В нашем примере это таблица «Кафедра».

Далее надо определить колонку таблицы-источника, значения из которой будут подставляться в описываемую колонку (рис. 5.13). В нашем примере таким полем является «Код_кафедры». Но так как пользователь вряд ли помнит коды, то желательно, чтобы при выборе нужного значения высвечивались названия кафедр. Для этого в окно «Выбранные поля» следует перенести еще и поле «Наименование_ кафедры_полное» или «Наименование_кафедры_краткое».

* Если вы изучаете данный учебник, выполняя на компьютере описываемые действия, то временно отложите выполнение шагов по созданию поля подстановки из другой таблицы. Завершите создание таблицы «Сотрудник», как описано далее; затем создайте таблицу «Кафедры», свяжите эти таблицы. После этого откройте таблицу «Сотрудник» в режиме **Конструктор** и выполните описанные здесь шаги.

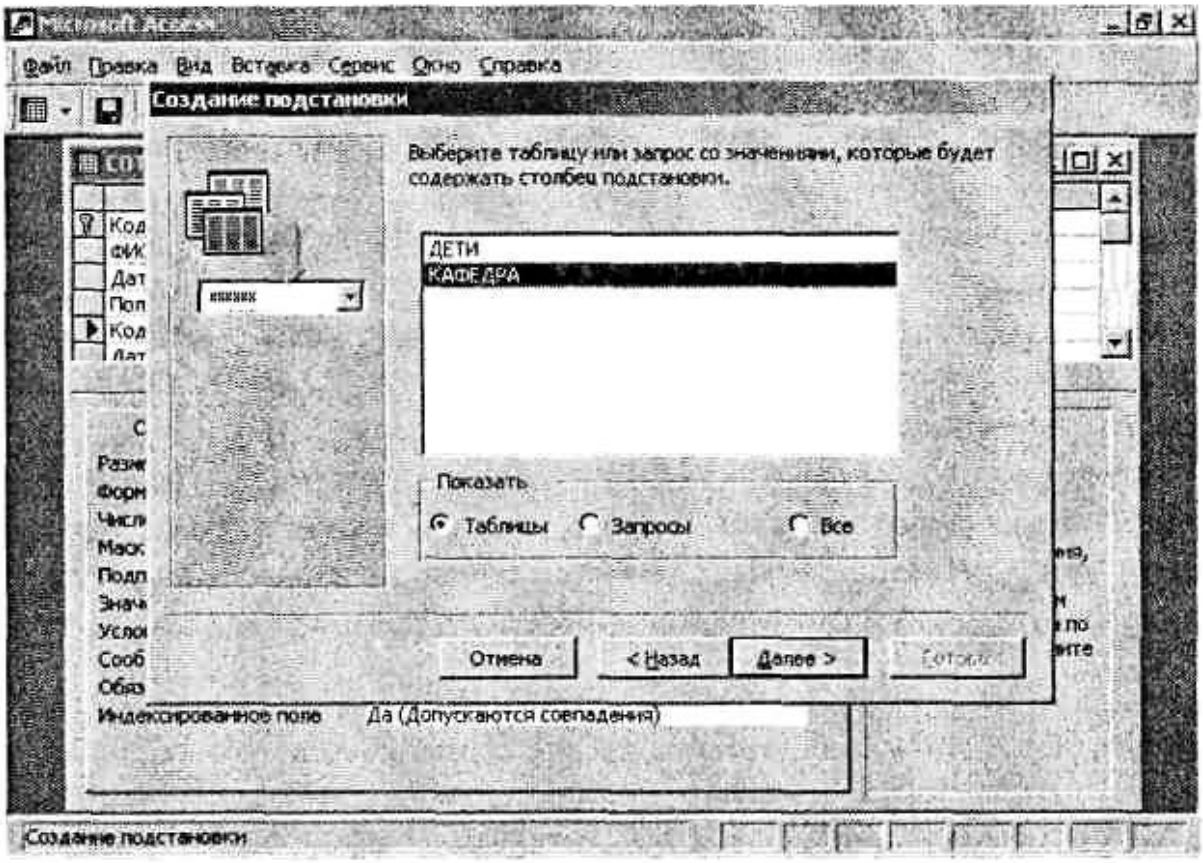


Рис. 5.12. Выбор таблицы/запроса источника для поля подстановки

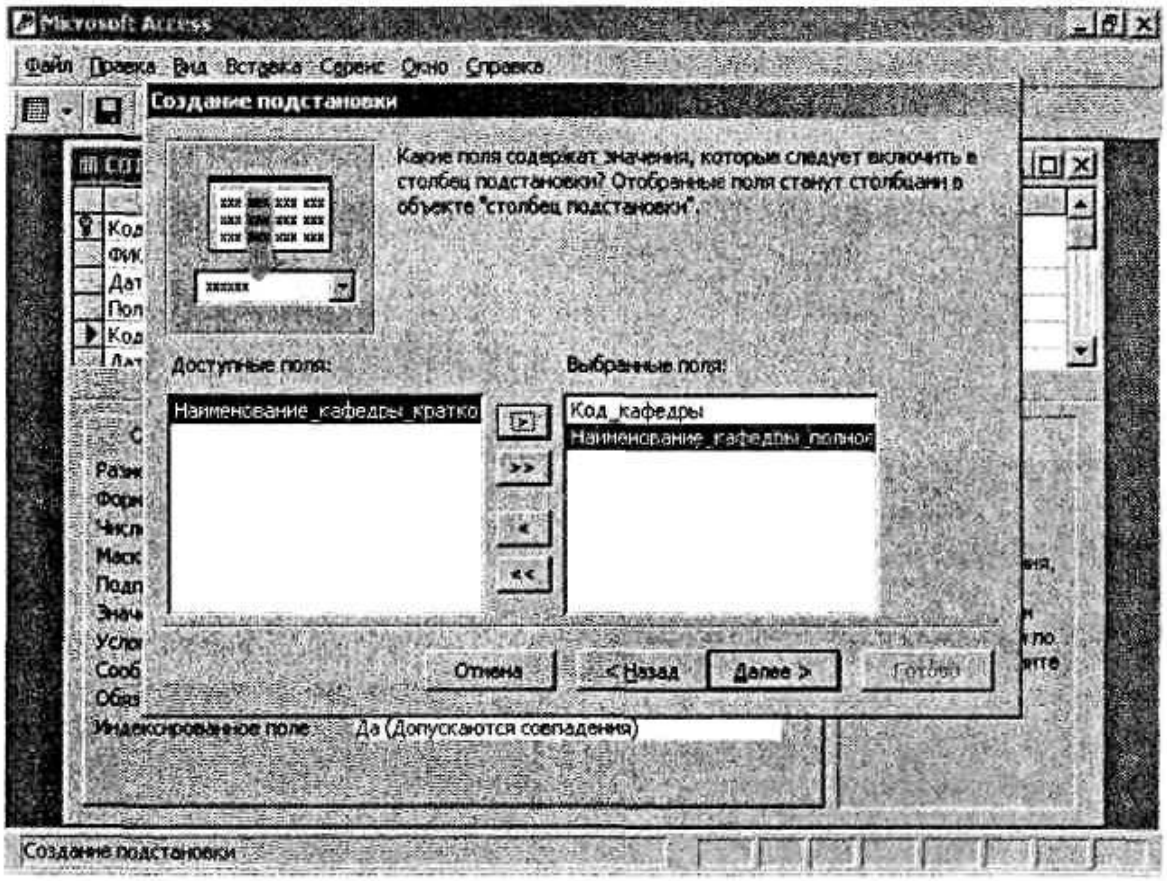


Рис. 5.13. Выбор колонки-источника для поля подстановки

В появившемся далее окне (рис. 5.14) можно не только задать ширину столбцов (позиционировавшись на границу столбца и перетащив ее в нужном направлении), но и определить, сколько столбцов будет выводиться на экран при вводе значения в это поле: если оставить знак «✓» в позиции «Скрыть ключевой столбец», то в нашем примере будет выводиться только поле «Наименование_кафедры_полное». Если эту «галочку» убрать, то будут выводиться оба поля: «Код_кафедры» и «Наименование_кафедры_полное».

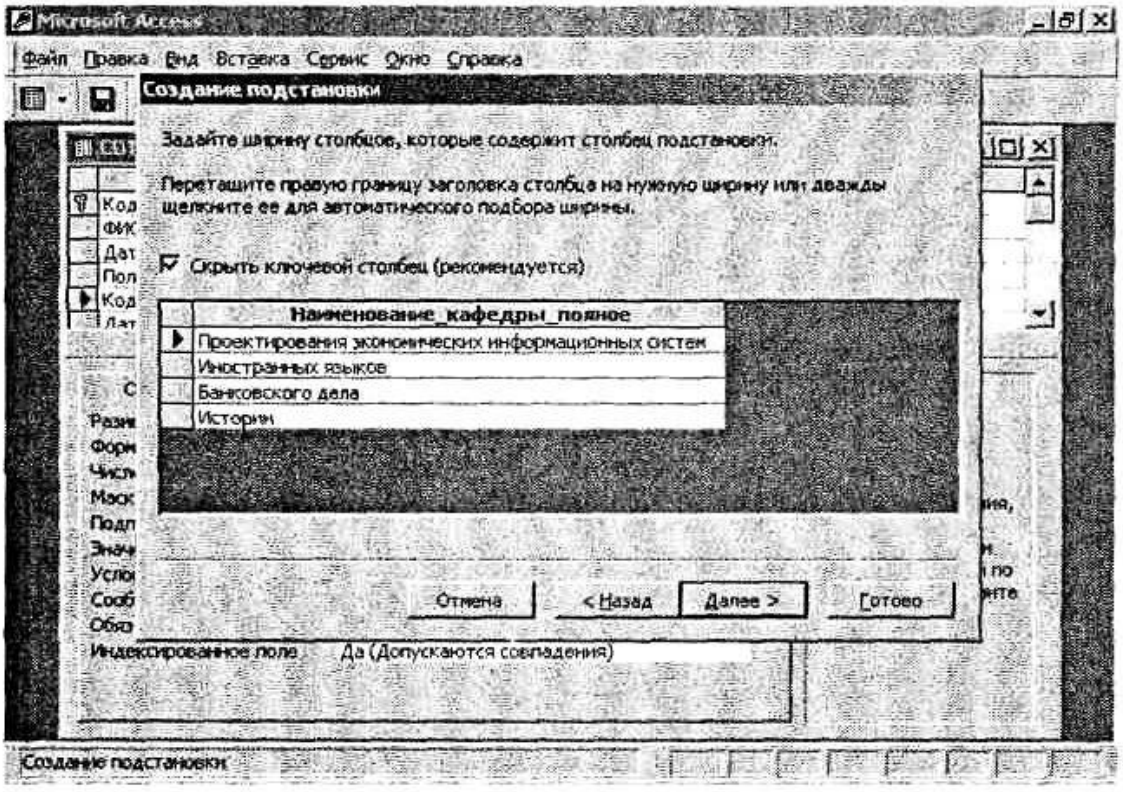


Рис. 5.14. Создание поля подстановки.
Задание ширины столбцов

Определение ключа таблицы

Каждая реляционная таблица по определению имеет ключ. Access позволяет задавать ключ при описании таблицы, но также разрешает и отказаться от этой возможности. По ключу система автоматически выполняет индексирование, а также проверяет уникальность значений ключа при вводе новых записей или их корректировке.

Если в качестве ключа выбрать автоматически задаваемый системой код (т.е. поле типа «счетчик»), то можно это поле первоначально не описывать, а подтвердить необходимость его создания при завершении описания таблицы (рис. 5.15). Access создаст это поле автоматически. Если после этого посмотреть описание структуры таблицы, то оно будет выглядеть так, как показано на рис. 5.16.

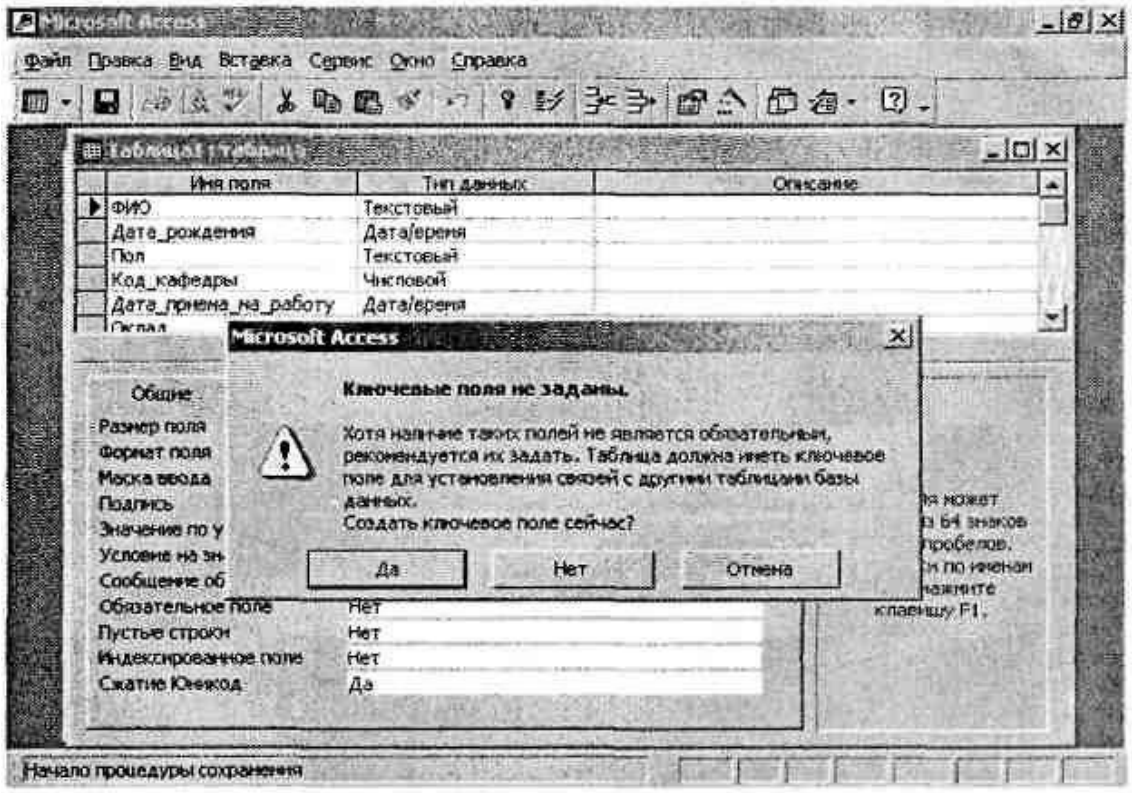


Рис. 5.15. Задание ключа таблицы в виде поля типа «счетчик» при закрытии таблицы после описания ее полей

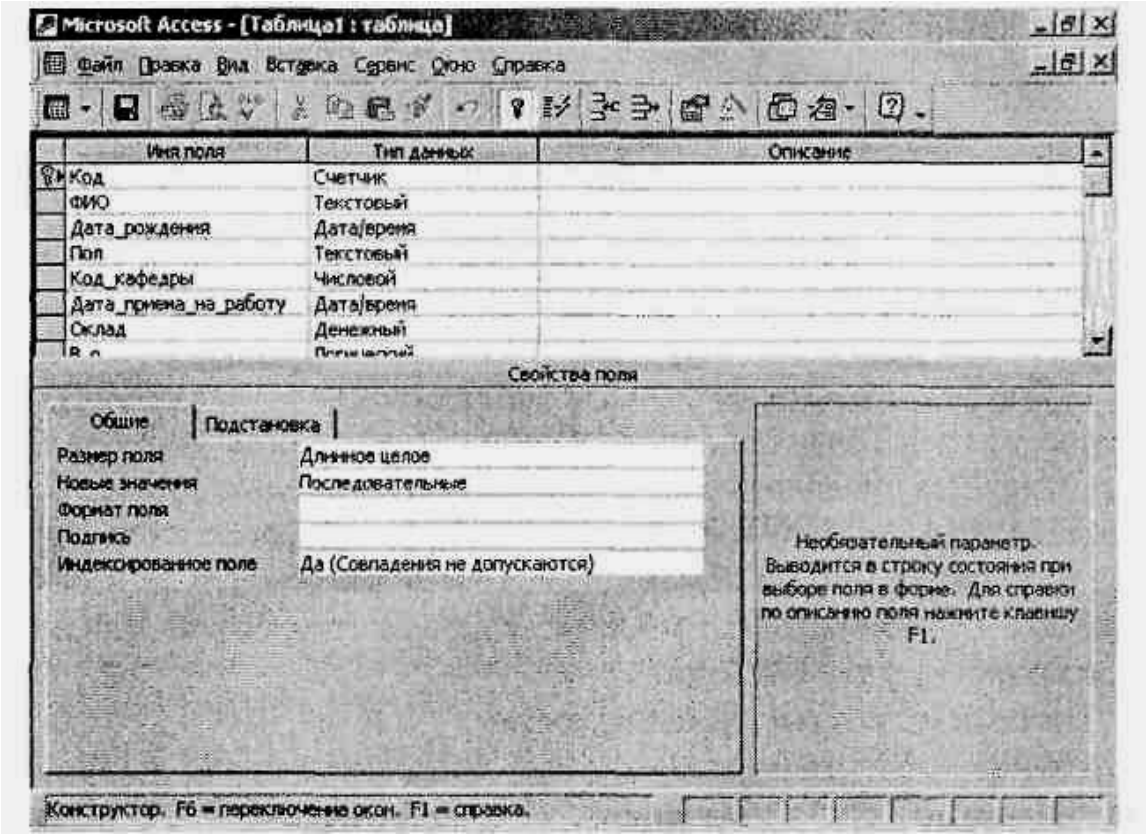


Рис. 5.16. Описание структуры таблицы с автоматически введенным системой ключевым полем «Код»

Access при создании ключа автоматически всегда называет это поле «Код». Для удобства дальнейшей работы переименуем ключевое поле в «Код_сотрудника» (рис. 5.17).

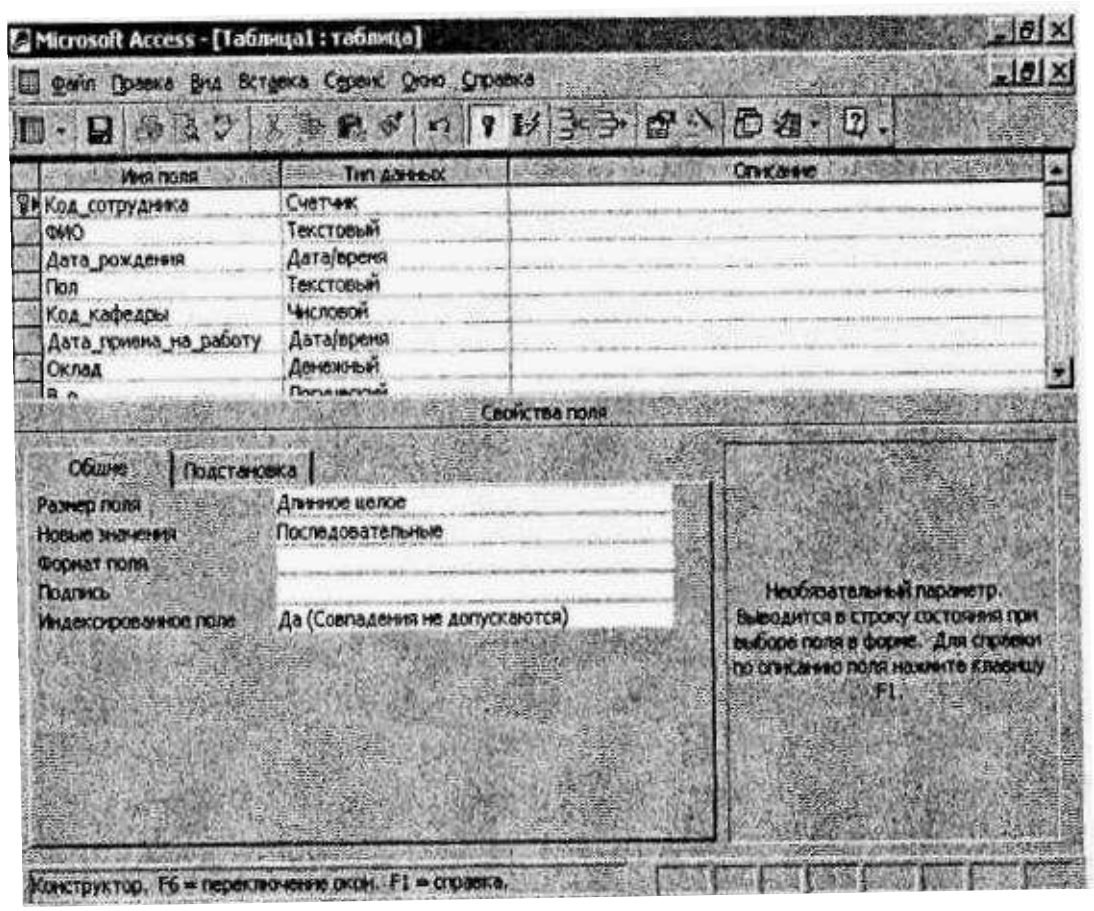


Рис. 5.17. Описание структуры таблицы с измененным ключевым полем

Если определять ключ самостоятельно, то это можно сделать несколькими путями: позиционироваться на соответствующее поле и щелкнуть по кнопке **Ключевое поле** () либо выбрать позицию меню **Правка/Ключевое поле**, либо воспользоваться правой кнопкой мыши для вызова контекстного меню, предварительно позиционировавшись на то поле, которое определяется как ключевое.

Свойства полей

В нижней части экрана описания таблицы отображается список свойств выбранного поля. Перечень свойств будет зависеть от выбранного типа поля (табл. 5.2).

Набор допустимых свойств вызывает некоторое удивление. Наверное, не всеми возможностями надо пользоваться. Так, обычно не рекомендуется проводить индексирование по логическому полю. Назначение поля МЕМО - хранение длинных текстов. Как и зачем задавать для них условия на значение

- не совсем понятно. То же (но несколько в меньшей степени) относится и к формату поля данного типа, а также формату поля Счетчик.

Поясним некоторые свойства полей.

Свойство «Индексированное поле» определяет, нужно ли создавать индекс по этому полю. Индекс ускоряет выполнение запросов, в которых используются индексированные поля, и операции сортировки и группировки.

Таблица 5.2

Свойство	Тип поля						
	Текстовый	Логический	МЕМО	Числовой	Дата/время	Денежный	Счетчик
Размер поля	+			+		+	+
Число десятичных знаков				+		+	
Формат поля	+	+	+	+	+	+	+
Маска ввода	+			+	+	+	
Подпись поля	+	+	+	+	+	+	+
Значение по умолчанию	+	+	+	+	+	+	
Условие на значение	+	+	+	+	+	+	
Сообщение об ошибке	+	+	+	+	+	+	
Обязательное поле	+	+	+	+	+	+	
Пустые строки	+		+		+		
Индексированное поле	+	+		+	+	+	+
Новые значения							

Значения свойства «Индексированное поле» приведены ниже.

Значение	**Описание**
Нет	(Значение по умолчанию). Индекс не создается
Да (допускаются совпадения)	В индексе допускаются повторяющиеся значения
Да (совпадения не допускаются)	Повторяющиеся значения в индексе не допускаются

В рассматриваемом примере в связи с тем, что по полю «ФИО» часто осуществляются поиск и упорядочение информации, желательно по нему провести индексацию. Поскольку среди сотрудников возможны однофамильцы, то должны быть разрешены совпадения значений индексируемого поля (рис. 5.18). Указанное по умолчанию значение свойства «Обязательное значение» «Нет» должно быть изменено на «Да».

Не допускается создание индексов для полей MEMO, гиперссылок и объектов OLE.

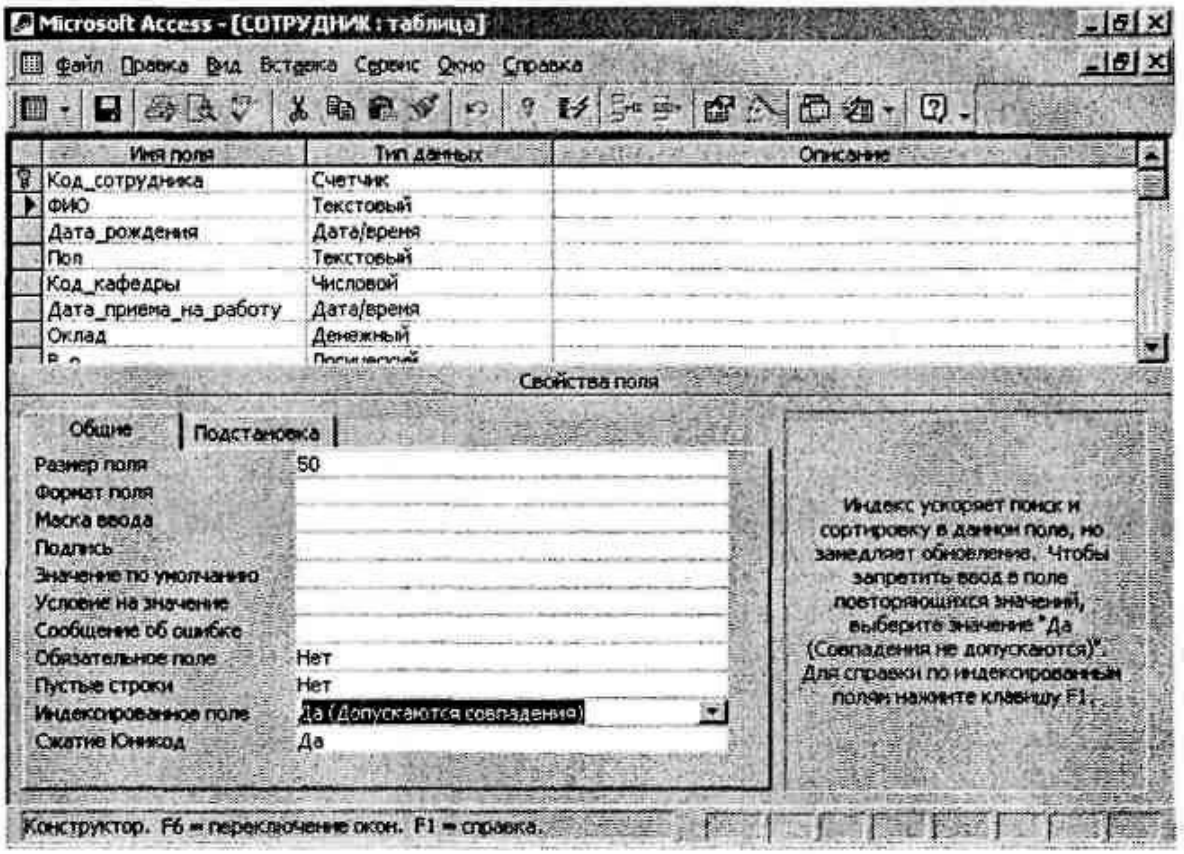


Рис. 5.18. Индексирование. Задание обязательности значений

Ключевое поле «Код», созданное системой автоматически, имеет тип «Счетчик» (*см.* рис. 5.16). Только для полей этого типа имеется свойство «Новые значения». Оно определяет способ увеличения значения поля счетчика при добавлении в таблицу новых записей.

Свойство «Новые значения» может иметь следующие значения:

- последовательные - значение поля счетчика увеличивается на 1 в каждой новой записи;
- случайные - поле счетчика в новой записи получает случайное значение типа Long Integer.

Следует отметить, что многие СУБД для полей такого типа позволяют использовать произвольный шаг приращения.

Свойство «Пустые строки» определяет, допускается ли ввод в данное поле пустых строк (строк, не содержащих символов). Значения этого свойства приведены ниже.

Значение	Описание
Да	Пустые строки являются допустимыми значениями
Нет	Пустые строки не являются допустимыми значениями

При задании значения «Да» для свойств «Пустые строки» и «Обязательное поле» Microsoft Access различает несуществующие данные (сохраняются в виде пустых строк) и данные, которые существуют, но не известны (сохраняются в виде пустых (Null) значений).

Для различия пустых строк от значений Null можно использовать свойство «Формат поля» (Format). При этом вместо пустых строк можно выводить строку «Отсутствуют данные».

В нашем примере значение поля «ФИО» должно присутствовать всегда и не может содержать пустых строк.

Сохранение описания таблицы

После того как описание таблицы завершено, его нужно сохранить. Этого можно достигнуть разными путями: выбрать меню **Файл/Сохранить** или **Файл/Закрыть** (после чего на вопрос «Сохранить изменения макета или структуры?» ответить «Да») или щелкнуть по кнопке **Вид** инструментального меню и выбрать **Режим таблицы** и на сообщение «Сначала необходимо сохранить таблицу. Сделать это сейчас?» ответить «Да» (этот способ надо использовать, если вы хотите сразу после описания структуры таблицы вводить данные в эту таблицу). В появившемся после указанных действий окне следует ввести имя созданной таблицы.

Создание таблиц для контрольного примера

Аналогичные действия повторяются при создании остальных таблиц БД.

Создадим таблицы «Кафедра» (рис. 5.19), и «Дети» (рис. 5.20).

При создании таблицы «Кафедра» ключ «Код_кафедры», как и в случае с таблицей «Сотрудник», создадим автоматически при закрытии таблицы.

При создании таблиц следует помнить, что в реляционных базах данных связывание таблиц происходит по значениям соответствующих полей связи. Эти поля должны соответствовать друг другу по типу и длине. В нашем примере речь идет о полях «Код_кафедры» в таблице «Кафедра» и в таблице «Сотрудник», и «Код_сотрудника» в таблицах «Сотрудник» и «Дети». Если в основной таблице ключевое поле имеет тип «Счетчик», то в подчиненной таблице соответствующее поле связи должно иметь тип «Числовой» и размер поля - «Длинное целое».

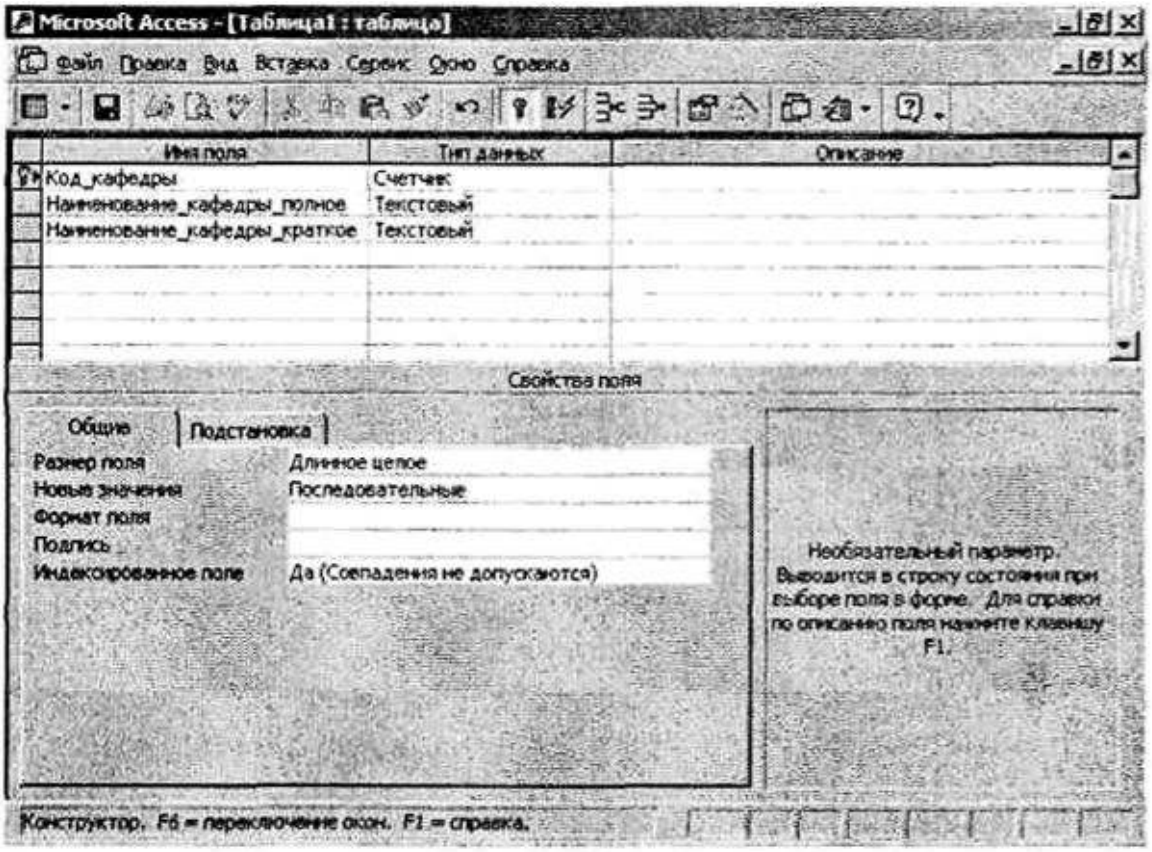


Рис. 5.19. Структура таблицы «Кафедра»

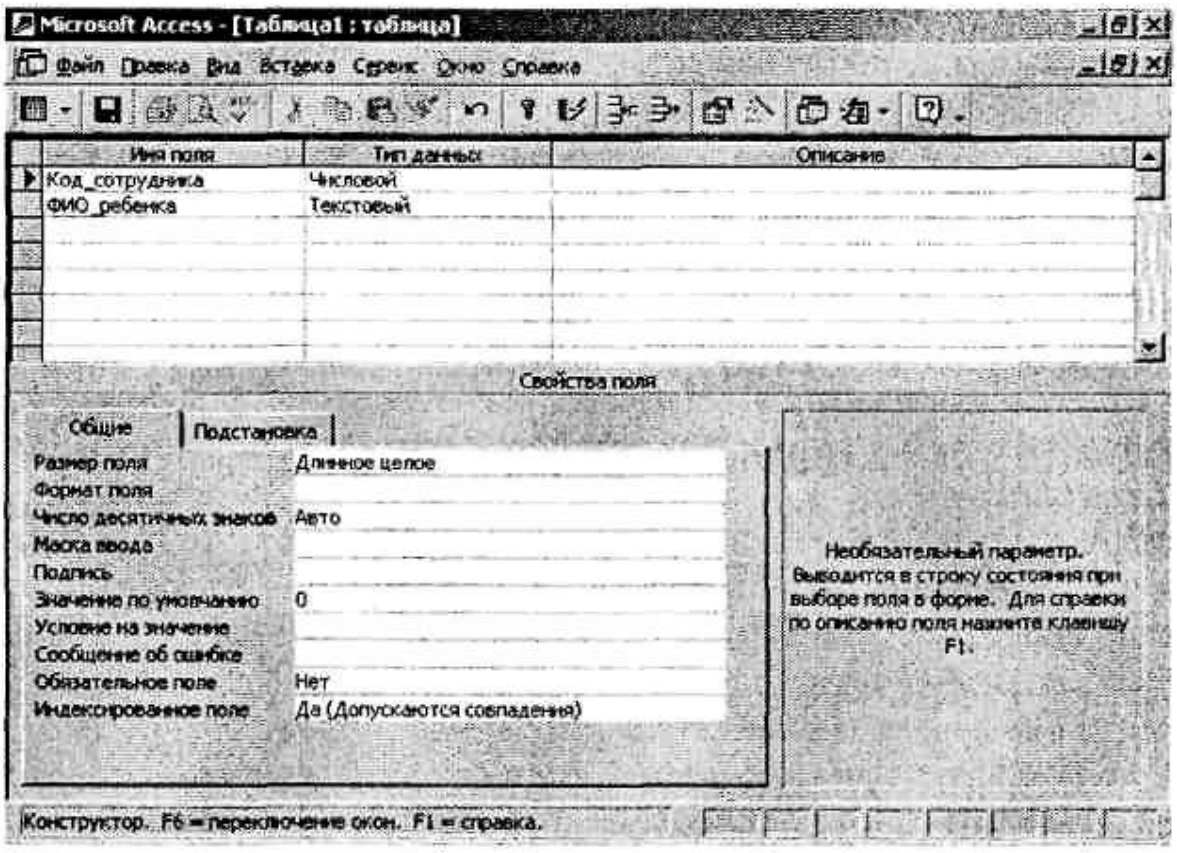


Рис. 5.20. Структура таблицы «Дети»

В таблице «Кафедра» для поля «Наименование_кафедры_краткое» создайте уникальный индекс.

При создании таблицы «Дети» воспользуемся возможностью вообще не задавать ключ. В реальной ситуации в таблицу «Дети» скорее всего будут введены еще и другие поля (например, дата рождения ребенка).

5.2.2. Изменение структуры таблиц

Если вы ошиблись при описании структуры таблицы или по каким-либо другим причинам хотите изменить ее, то это можно легко сделать. Если вы уже вышли из процесса создания таблицы, но еще продолжаете работать с ней, то можно перейти обратно в режим **Конструктор,** воспользовавшись кнопкой **Вид**. Если нужная таблица вообще закрыта, то ее можно открыть в режиме **Конструктор** и таким образом вернуться в окно описания таблицы.

Для добавления поля в таблицу выберите строку, над которой требуется добавить новое поле, и щелкните по кнопке **Добавить строки** на панели инструментов либо просто нажмите клавишу [Ins]*. Для добавления поля в конец таблицы выберите первую пустую строку и вводите в нее описание очередного поля.

Если таблица уже содержит данные, то рекомендуется до изменения типов данных и размеров полей сделать ее копию, поскольку несовместимость существующих данных с новым значением свойства «Тип данных» может привести к потере данных.

5.2.3. Другие способы создания таблиц

Если создается таблица, структура которой имеет много общего со структурой ранее созданной таблицы, то можно скопировать структуру существующей таблицы (для этого нужно позиционироваться на соответствующей таблице, выбрать позицию меню **Правка/Копировать,** затем - **Правка/Вставить,** после чего в появившемся окне (рис. 5.21) ввести имя вновь создаваемой таблицы, а в качестве параметра вставки выбрать «только структура»). Структура созданной таким образом таблицы может быть впоследствии скорректирована обычным способом. В приведенном примере в базе данных учебного заведения на основе таблицы «Сотрудник» строится таблица «Аспирант».

* При работе с Access (как, впрочем, и с другими Windows-системами) рекомендуется активно пользоваться правой кнопкой мыши для вызова контекстного меню. Часто это бывает самым простым путем выбора нужного действия.

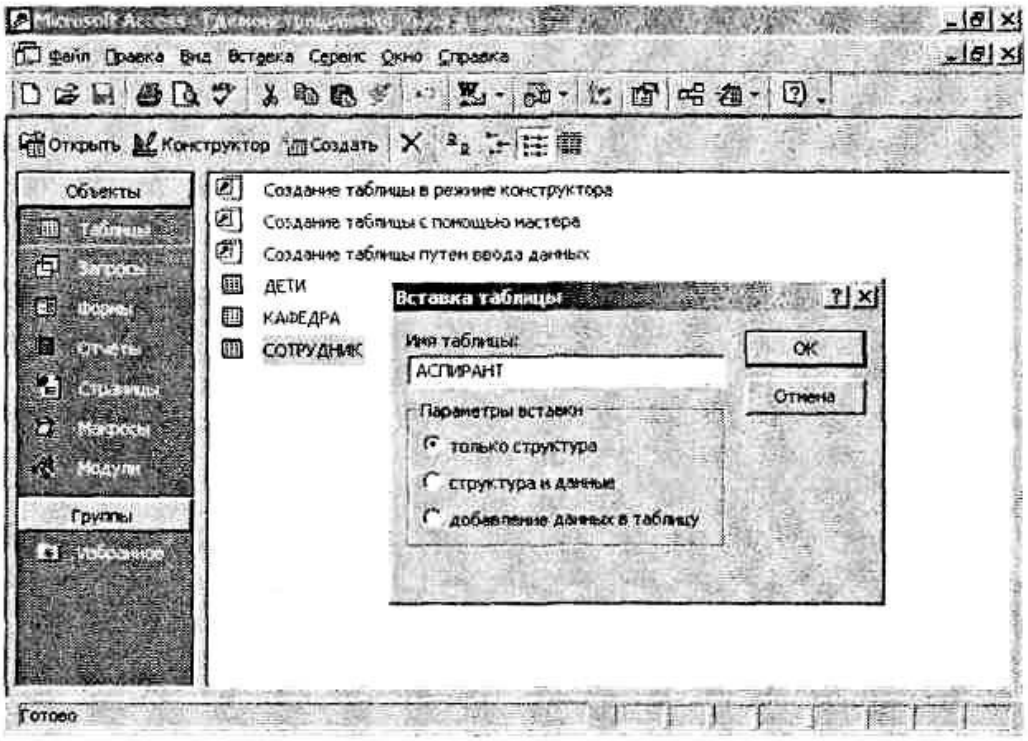


Рис. 5.21. Копирование структуры таблицы

Кроме рассмотренных выше способов создать таблицу можно с использованием Мастера таблиц (**Таблица/Создать/Мастер таблиц**). В левой части окна **Создание таблиц** высвечивается перечень образцов таблиц, из которых можно выбрать подходящую по содержанию таблицу (рис. 5.22).

Из выбранной таблицы можно перенести все или некоторые поля во вновь создаваемую таблицу, можно изменить имя поля. Чтобы ввести какие-либо другие изменения в структуру создаваемой таблицы, следует завершить формирование таблицы с помощью Мастера, после чего откорректировать структуру в обычном порядке. Как видим, использование этой возможности не освобождает от понимания основ проектирования БД, так как нужно оценить, насколько предлагаемое в качестве образца решение соответствует вашим потребностям, и при необходимости изменить предлагаемую структуру БД.

Создать таблицу можно и путем импорта ее из других систем. Кроме того, в виде таблицы можно сохранить результат запроса (см. главу 6). В Access имеется еще возможность создавать таблицу в режиме таблицы, но этот способ не представляется интересным.

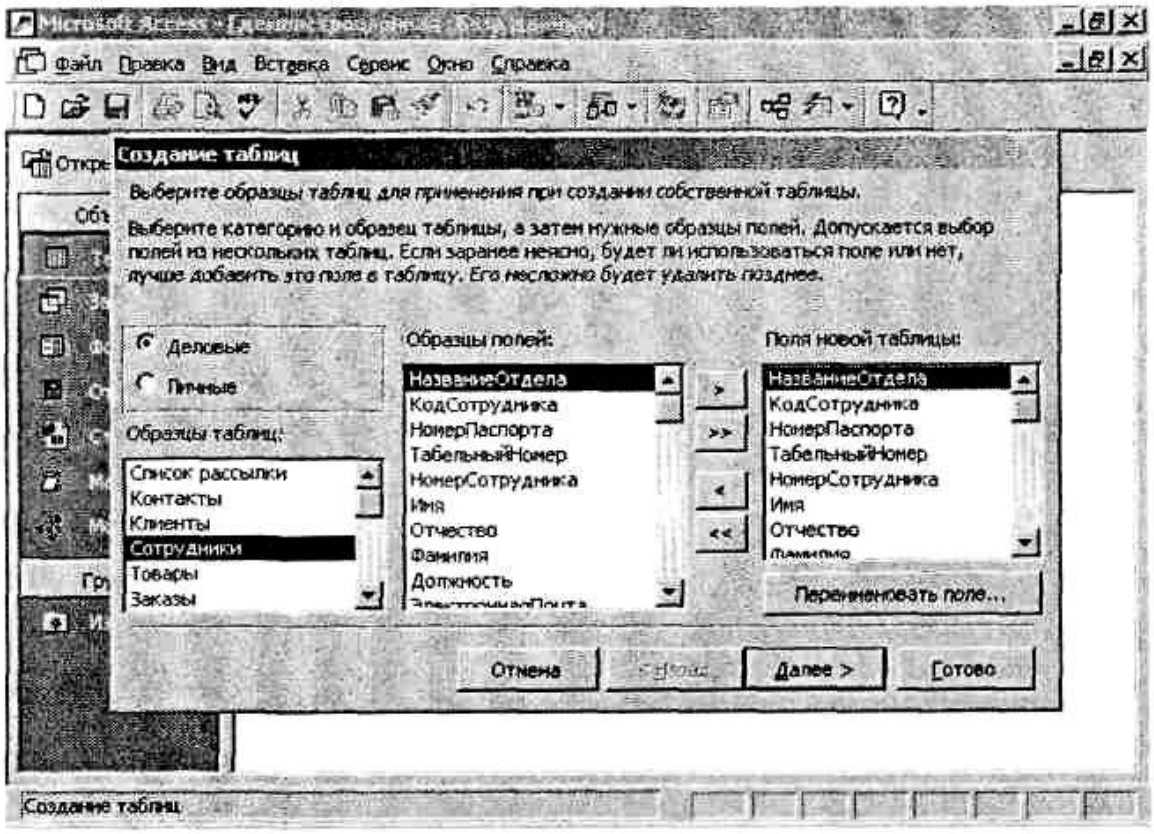


Рис. 5.22. Создание таблиц с использованием Мастера таблиц

5.2.4. Связывание таблиц

После того как таблицы созданы, можно задать их связанность. Для этого надо выбрать позицию меню **Сервис/Схема данных** (либо щелкнуть по соответствующей кнопке на панели инструментов). Далее в открывшемся окне **Схема данных** следует добавить в окно те таблицы, между которыми будет определяться связь. Такие таблицы чаще всего связаны отношением 1:M. Для установления связи нужно позиционироваться на поле связи (обычно это первичный ключ) в основной таблице (та, которая стоит на стороне «1»), не отпуская клавишу мыши, перетащить появившийся значок на соответствующее поле в зависимом файле и отпустить клавишу мыши. После этого на экране появится окно **Изменение связей** (рис. 5.23). Далее следует определить, нужно ли задавать ограничения целостности связи, и если да, то выбрать режимы корректировок (обновления и удаления). Если вы задаете ограничения целостности, то поле связи основной записи должно быть проиндексировано.

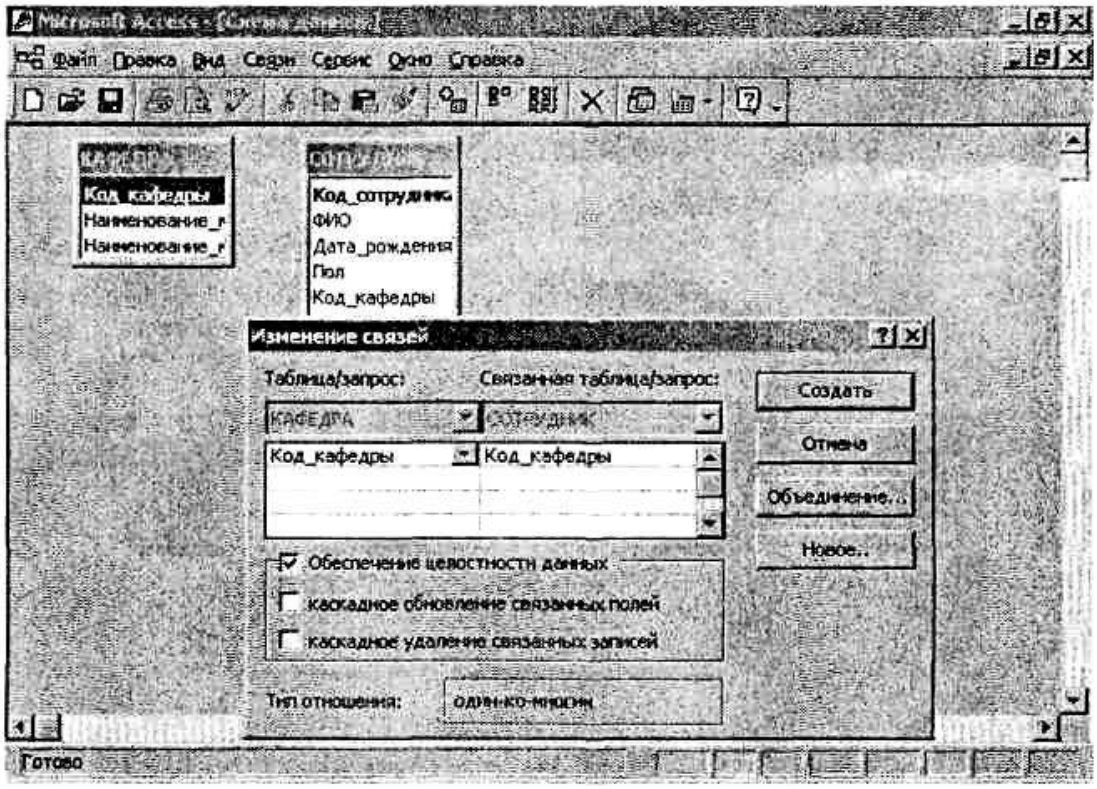


Рис. 5.23. Задание связи и ограничений целостности по связи

В рассматриваемом в качестве примера случае при связывании таблиц «Кафедра» и «Сотрудник» ограничение целостности следует задать, чтобы в таблице «Сотрудник» не появлялись коды кафедр, которые отсутствуют в справочнике «Кафедра». Задавать «каскадное обновление связанных полей» в данном случае не имеет смысла. Каскадное обновление означает, что при изменении первичного ключа в основной таблице соответствующие поля в связанной таблице автоматически изменяются. В таблице «Кафедра» поле «Код_кафедры» имеет тип «Счетчик». Это означает, что изменять значение этого поля нельзя. Задавать каскадное удаление в данном случае тоже опасно, поскольку в случае ликвидации какой-то кафедры и удаления соответствующей записи из таблицы «Кафедра» окажутся удаленными из таблицы «Сотрудник» все записи сотрудников, работавших на этой кафедре.

Для связи же таблиц «Сотрудник» и «Дети» (рис. 5.24) каскадное удаление вполне уместно.

Существуют понятия *внутреннего*, *левого* и *правого* соединения. В окне **Изменение связей**, появляющемся при установлении связи между двумя таблицами, есть кнопка **Объединение**, щелкнув по которой пользователь попадает в окно **Параметры объединения**, в котором он может выбрать один из трех вариантов:

1. объединение только тех з*апи*сей, в которых значения связанных полей обеих таблиц совпадают;
2. объединение всех записей первой таблицы и только тех записей из второй таблицы, в которых значения связанных полей обеих таблиц сов-

падают;

3. объединение всех записей второй таблицы и только тех записей из первой таблицы, в которых значения связанных полей обеих таблиц совпадают.

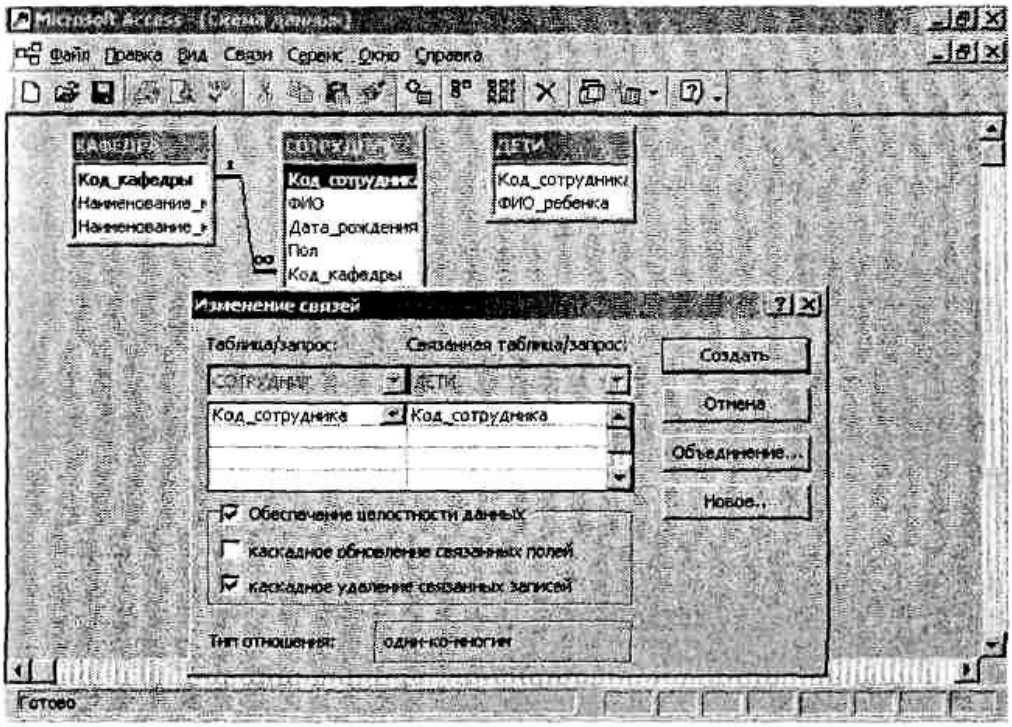


Рис. 5.24. Задание целостности связи и условий обновления связанных записей

Первый из перечисленных вариантов обозначает *внутреннее*, второй - *левое*, третий - *правое соединение*.

Параметры объединения влияют на то, какие записи из связанных таблиц будут выводиться в ответ при просмотре базы данных. Поэтому при реализации запросов следует обязательно иметь в виду, как связаны таблицы, и при необходимости изменять параметры объединения. Более подробно эти аспекты будут изложены при рассмотрении языков запросов (см. главы 6 и 7).

5.2.5. Просмотр связанных таблиц

В Access 2000 и выше при работе с таблицей можно просматривать и корректировать данные не только текущей таблицы, но и связанных с ней таблиц (подтаблиц).

Предположим, что имеется база данных со схемой, представленной на рис. 5.25.

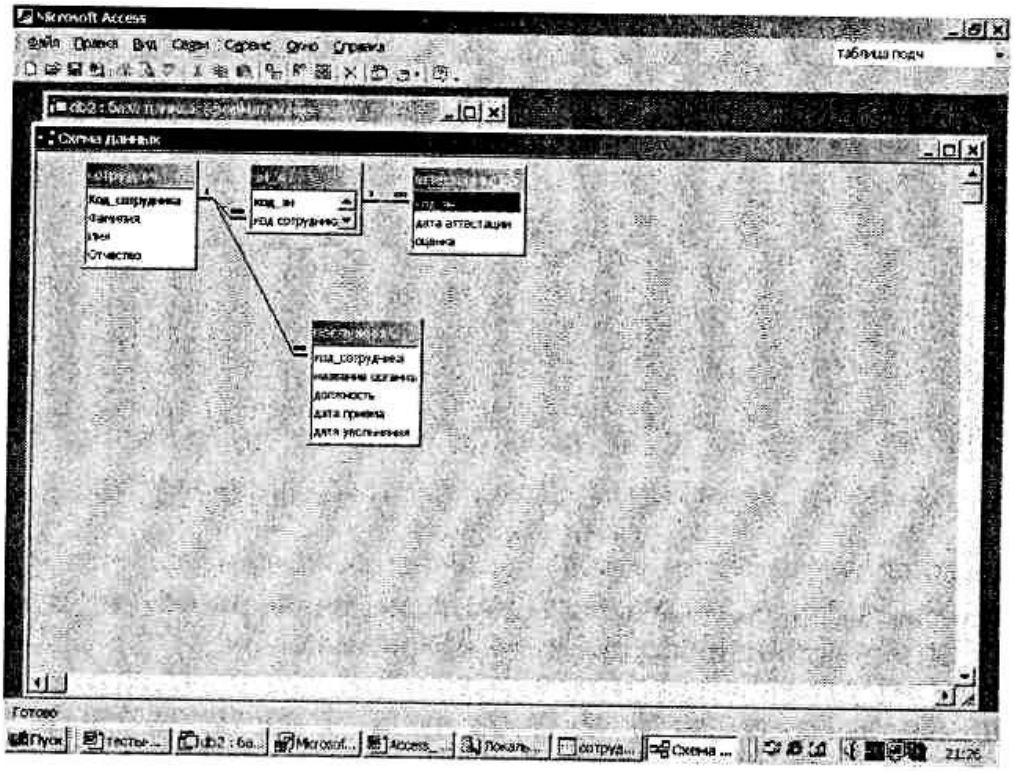


Рис. 5.25. Схема базы данных

Если таблицы предварительно связаны, то при просмотре таблицы, имеющей подчиненные ей таблицы, между колонкой выделения и первым столбцом таблицы появляется столбец, в котором в каждой строке стоит знак «+» (рис. 5.26). Если щелкнуть по нему левой клавишей мыши, то высвечиваются те записи подчиненной таблицы, которые относятся к данной записи основной таблицы (рис. 5.27).

Если таблица имеет несколько непосредственно подчиненных ей таблиц, то одновременно в режиме таблицы можно просматривать только одну из них. Определить, какая это будет подтаблица, можно, если выбрать позицию меню **Вставка/Подтаблица** и в открывшемся окне **Вставка подтаблицы** - нужную подтаблицу (рис. 5.28).

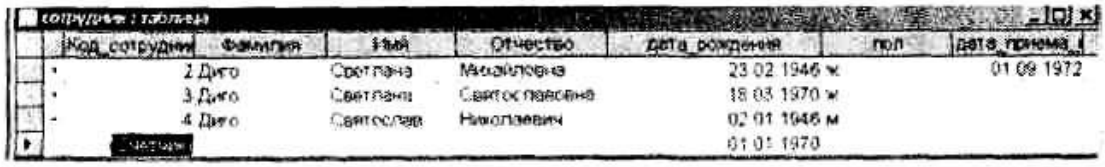


Рис. 5.26. Просмотр связанных таблиц (экран 1)

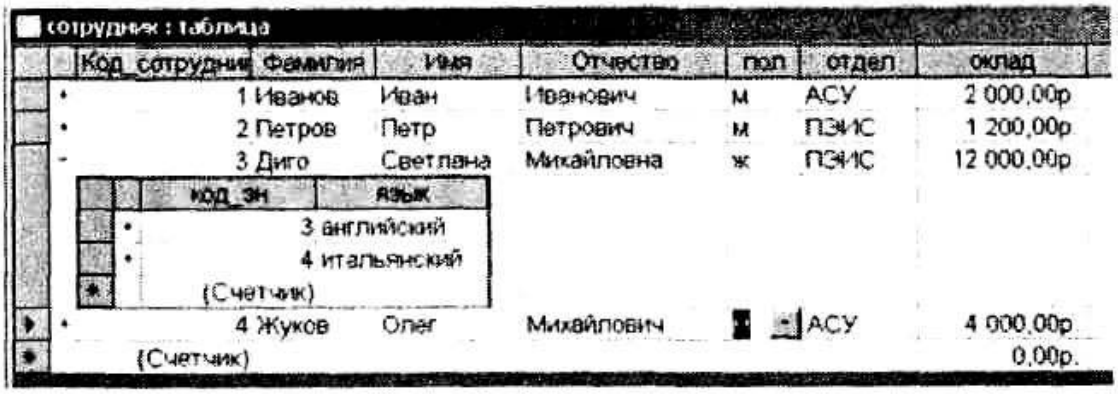


Рис. 5.27. Просмотр связанных таблиц (экран 2)

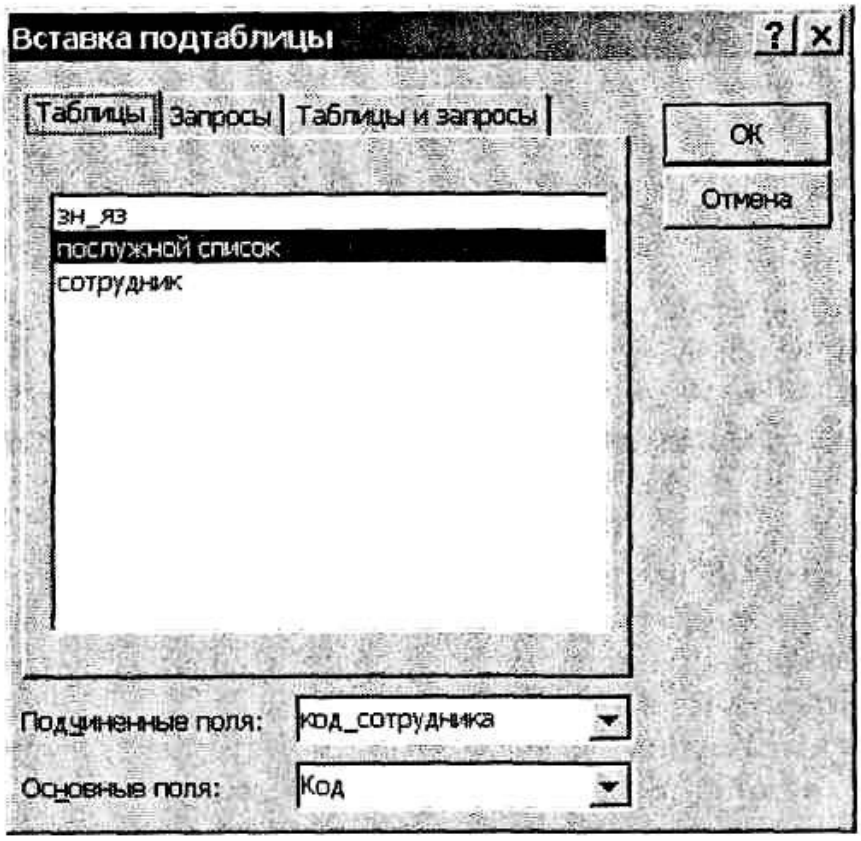


Рис. 5.28. Выбор подтаблицы

В то же время можно просматривать цепочку иерархически связанных подтаблиц. Если имеется последовательность иерархически связанных таблиц (на рис. 5.25 это «Сотрудник»-«Зн_яз»-«Аттестации_яз»), то окна взаи-

мосвязанных таблиц могут быть открыты для всей цепочки взаимосвязанных таблиц (рис. 5.29).

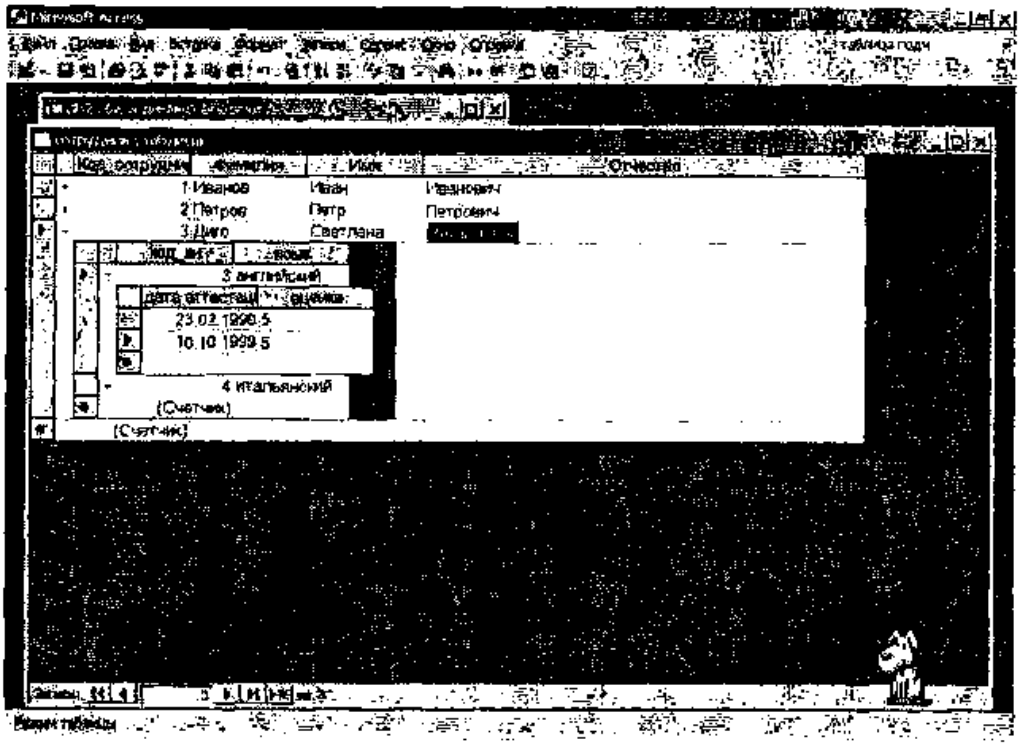


Рис. 5.29. Просмотр цепочки взаимосвязанных таблиц

5.2.6. Задание ограничений целостности в Access

Обеспечение целостности БД - важнейшая задача при создании БнД, поскольку обеспечение адекватности базы данных отображаемой предметной области является одним из основных требований, предъявляемых к БнД.

При изложении вопросов создания и связывания таблиц мы уже касались некоторых аспектов обеспечения целостности БД. Рассмотрим другие возможности задания ограничений целостности.

В Access многие ограничения целостности могут задаваться при создании таблицы.

Ограничения, относящиеся к полю

Тип поля. Он определяет допустимые символы, которые могут быть использованы при его заполнении (в частности, не допускается ввод текста в числовые поля).

Для некоторых типов полей, например поля типа «дата», осуществляется и более сложная проверка. Если допущена ошибка в типе данных или неправильно введена дата, то пользователь должен обязательно исправить ошибку, так как СУБД не дает других возможностей продолжить работу.

Ряд свойств полей также позволяет обеспечивать контроль целостности:

- размер поля;

- формат поля;
- маска ввода;
- значение по умолчанию;
- условия на значения;
- сообщение об ошибке;
- обязательное поле;
- пустые строки;
- индексированное поле.

Каждое из них в той или иной степени связано с ограничениями целостности.

Поясним использование некоторых из перечисленных выше свойств в целях обеспечения контроля целостности на отдельных примерах.

Размер поля. В поле нельзя ввести больше символов, чем это зафиксировано в свойстве «размер поля» или предопределено типом поля.

Условия на значения. Это одна из самых гибких возможностей определения ограничений целостности. Условия вводятся как выражения. Выражения могут быть простыми или сложными. Используя их, можно задавать и диапазоны. Например, условие: >#1.92#, заданное как «Условие на значения» для поля «Дата_приема_на_работу», будет означать, что допустим ввод дат только после 1992 г. Значения даты необходимо заключать в символы номера (#). Такое ограничение целостности может быть использовано, например, в случае, если организация, для которой ведется БД, была создана 1 января 1992 г. и все зачисления на работу были после этой даты. При задании такого ограничения целостности ввод значения в поле будет обязательным (даже если в свойстве поля «Обязательное поле» зафиксировано -«Нет»).

Условия на значения могут задаваться для полей или записей. Выражения, определяющие условия на значения, не должны содержать функции, определяемые пользователем, статистические функции или функции по подмножеству, функции CurrentUser или Eval, а также ссылки на формы, запросы и таблицы. Кроме того, выражение, указанное в качестве условия для поля, не должно содержать ссылки на другие поля.

Выражение, указанное в качестве условия на значение для записи, может содержать ссылки на поля той же таблицы. Условия на значения для записей задаются в окне свойств таблицы, вызываемом командой **Свойства** меню **Вид** в режиме **Конструктор таблицы**.

Если пользователь задает значение свойства «Условие на значение», но не определяет свойство «Сообщение об ошибке», то при нарушении условия на значение Microsoft Access выводит стандартное сообщение об ошибке. Если значение свойства «Сообщение об ошибке» задано (рис. 5.30), то в сообщении об ошибке выводится текст, указанный в качестве значения этого свойства.

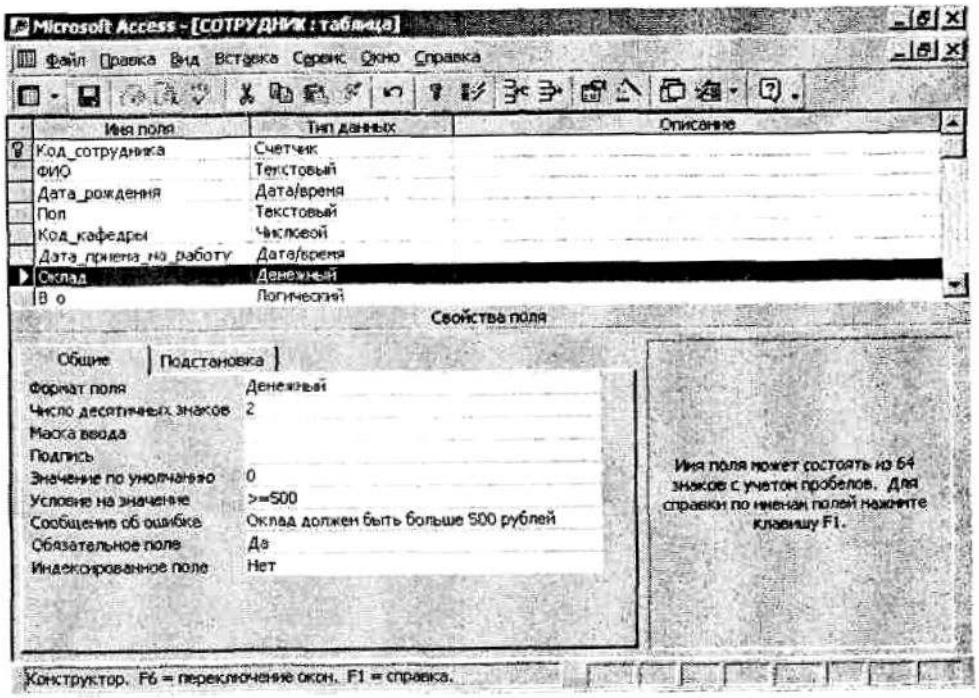


Рис. 5.30. Задание ограничений целостности при описании полей таблицы

В Access нет специального способа задания домена перечислением. Как было показано выше, этого можно достичь, используя «Мастер подстановки». Кроме того, это можно сделать и путем задания соответствующего выражения для свойства «Условие на значение». Например, для поля «Должность» в БД сотрудников вузов можно задать условие:

«ассистент» OR «старший преподаватель» OR
«доцент» OR «профессор».

На рис. 5.31 приведен пример задания ограничения на домен для поля «Пол». Кроме того, для этого поля задано и «Значение по умолчанию».

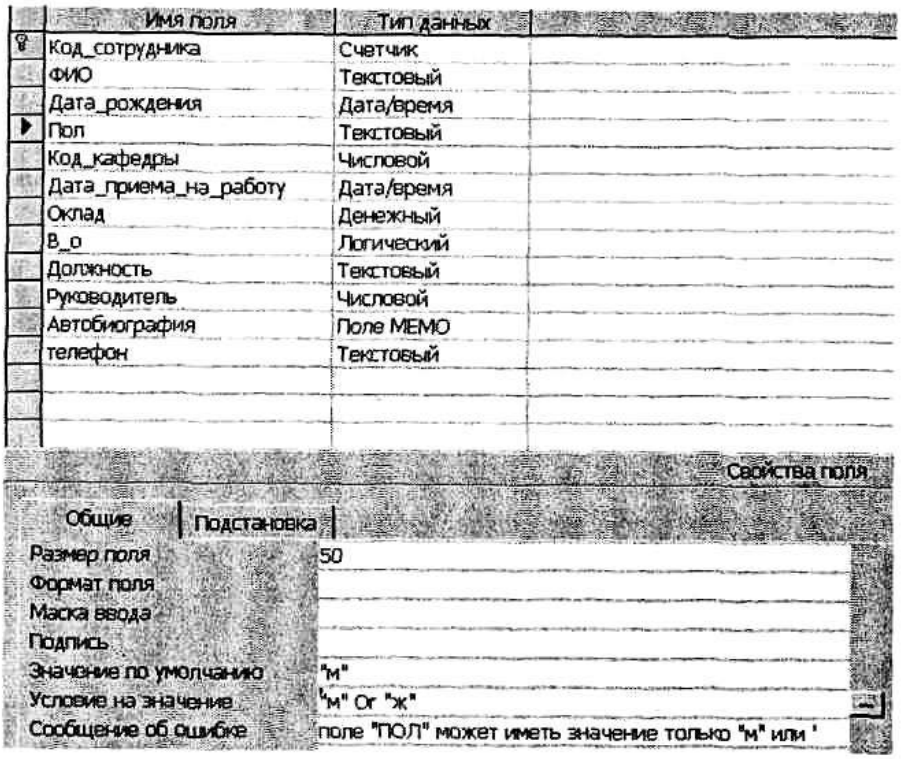


Рис. 5.31. Задание ограничения на домен

Приведенный на рис. 5.30 пример демонстрирует задание открытого диапазона. Закрытый диапазон для контроля введенных значений приведен на рис. 5.32.

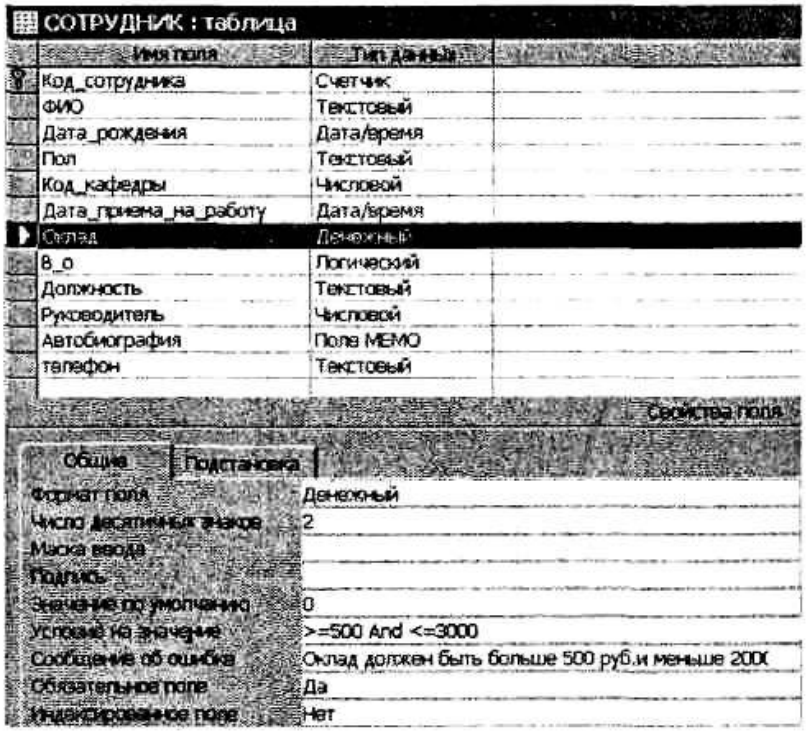


Рис. 5.32. Задание закрытого диапазона (вариант 1)

Это же ограничение может быть задано и иначе (рис. 5.33), что более удобно.

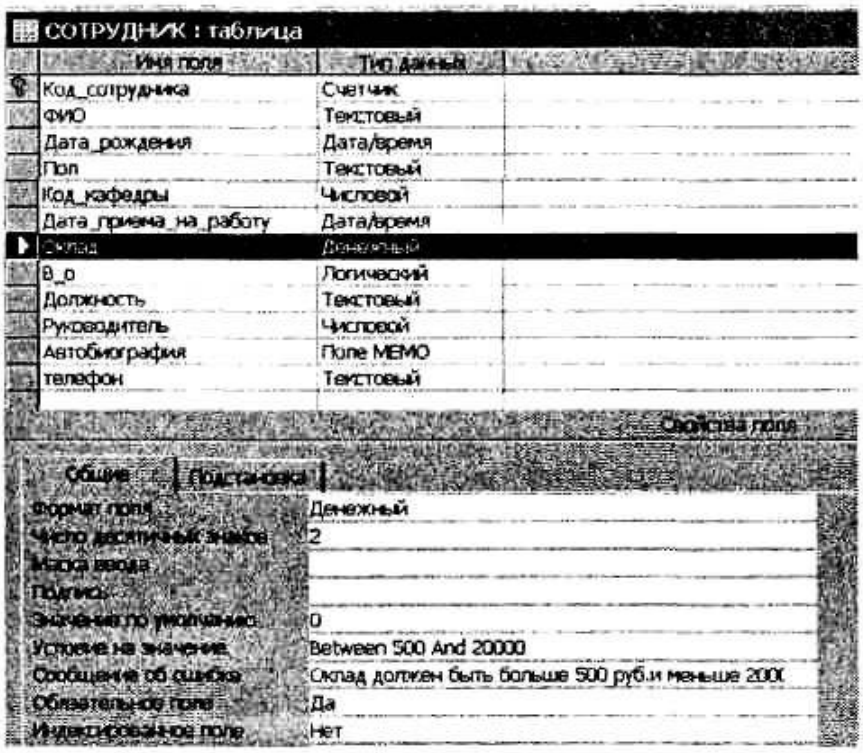


Рис. 5.33. Задание закрытого диапазона (вариант 2)

Маска ввода. Она применяется для того, чтобы ограничить символы, которые будут использованы при заполнении значения поля, и это может рассматриваться как один из способов задания ограничений целостности.

Предположим, нужно ввести в таблицу имена сотрудников. Для соответствующего поля можно задать маску ввода, которая позволит использовать только буквы при вводе, обеспечит преобразование первого символа в верхний регистр, всех остальных - в нижний, и допускает использование не менее двух букв (считаем, что имен, состоящих из одной буквы, нет).

В Access такая маска ввода будет выглядеть следующим образом:

>L<L??????????????????????????

Символ «L» в маске обозначает, что в данную позицию должна быть введена буква, символ «?» - что в данную позицию может быть введена буква. Символ «>» преобразует все символы, расположенные правее этого знака, к верхнему регистру, символ «<» преобразует все символы, расположенные правее этого знака, к нижнему регистру.

Все символы, допустимые в масках, и их назначение можно посмотреть в «Справочной системе» Access.

Маска ввода применяется только при вводе знаков с клавиатуры и игнорируется при всех остальных операциях.

Маски ввода не только обеспечивают контроль допустимости используемых символов, но и облегчают процесс ввода данных.

Индексированное поле. Его можно использовать для контроля на уникальность. Для этих целей следует задать свойство «Уникальный индекс».

В тех СУБД, которые поддерживают концепцию ключа (в том числе и в Microsoft Access), после того как в таблице определяется ключ, по этому полю проводится индексирование и запрещается ввод повторяющихся или пустых значений ключа.

Ограничения, относящиеся к записи

Если ограничения целостности затрагивают несколько полей одной записи, то они должны задаваться не как свойство записи, а как свойство таблицы (рис. 5.34). Чтобы попасть в окно **Свойства таблицы,** надо, находясь в окне конструктора таблицы, позиционироваться на заголовке этого окна, нажать правую клавишу мыши и в появившемся контекстном меню выбрать позицию **Свойства.**



Рис. 5.34. Задание ограничений целостности на запись

Целостность связи

Как отмечалось выше, при задании связи между таблицами в Access можно установить флажок **Поддержание целостности данных**, и в этом случае система будет автоматически поддерживать ограничения целостности связи.

При удалении основной записи, связанной с несколькими подчиненными, могут быть выбраны разные стратегии обновления:

1. запретить удалять основную запись, если имеются подчиненные;
2. удалить вместе с основной записью и все подчиненные (каскадное уда-

ление).

В Access при поддержании целостности связи автоматически принимается первая стратегия. Чтобы отменить ее для данной связи, надо установить флажок **Каскадное удаление связанных записей**.

Если установлено **Поддержание целостности данных** и не установлено **Каскадное удаление связанных записей**, то из основной таблицы нельзя будет удалить запись, если в зависимой таблице есть связанные с ней записи.

Обеспечить ссылочную целостность можно и иным способом -используя поле подстановки: если значения будут переноситься из связанной таблицы, то в подчиненной не может появиться значение, отсутствующее в основной таблице.

Для задания сложных условий проверки целостности можно использовать макросы или модули.

5.3. Организация ввода и корректировки данных в БД

5.3.1. Общие сведения

После того как завершено проектирование структуры базы данных и БД описана на ЯОД, можно приступать к вводу данных. Это можно сделать как сразу по окончании описания структуры таблицы (или иной информационной единицы, если речь идет не о реляционной СУБД), так и потом.

Если для осуществления ввода просто воспользоваться командой ввода записи, то обычно на экране появляется структура записи в следующем виде: все поля размещаются одно под другим (анкетная форма) в том порядке, в котором они были заданы при описании структуры файла; для каждого поля слева указывается его идентификатор, а справа имеется окошко, длина которого совпадает с длиной поля. В это окошко можно вводить требуемое значение поля.

Другой способ - идентификаторы полей размещаются друг за другом слева направо, а окошки располагаются под ними (табличная форма). Такой способ ввода данных имеет много очевидных недостатков.

1. Идентификаторы полей часто бывают короткими и малоинформативными.
2. Порядок ввода полей может не совпадать с порядком их расположения в структуре файла БД.
3. Нерационально используется пространство экрана.
4. Не все СУБД позволяют задать контроль ограничений целостности при описании структуры БД и обеспечивать таким образом контроль правильности ввода.
5. Оформление экрана оставляет желать лучшего.
6. Невозможно в полной мере управлять выдачей подсказок пользователю.

Чтобы устранить эти недостатки, для организации ввода данных обычно создаются специальные экранные формы. СУБД обладают разными возмож-

ностями для создания экранных форм. Большинство современных систем имеет в своем составе специальные генераторы экранных форм, обеспечивающие конечному пользователю без программирования создание экранных форм требуемого вида. Кроме того, генераторы экранных форм обычно позволяют без программирования задавать многие виды ограничений целостности.

В СУБД, не имеющих соответствующих генераторов, экранные формы и процедуры проверки правильности данных при вводе должны программироваться с использованием иных языковых средств системы.

Поскольку экранные формы могут использоваться для организации как ввода, так и вывода информации, этот вопрос выделен в отдельный раздел учебника (см. главу 8).

Заполнение БД можно осуществлять посредством ввода данных с клавиатуры, переноса данных из других файлов (той же СУБД) или путем импорта из других систем. Кроме того, можно формировать запись или отдельные ее элементы программным путем.

При организации ведения баз данных нужно стремиться к реализации принципа однократного ввода информации. Вводить вручную информацию в БД следует только тогда, когда невозможно это сделать иными способами, используя уже имеющуюся в ИС информацию. Современные СУБД предоставляют простые и быстро реализуемые возможности по связыванию и совместной обработке файлов, по переносу данных из одних файлов в другие, возможности экспорта/импорта в другие системы; и этим надо пользоваться, избегая повторного ручного ввода данных. Кроме того, нужно стремиться к достижению минимально возможного уровня дублирования данных.

При организации ввода данных в базу данных надо стараться сокращать обьем данных, вводимых пользователем с клавиатуры. Наряду с уменьшением затрат времени на ввод данных это приводит и к сокращению ошибок при вводе. Сокращения объема вводимых с клавиатуры данных можно добиться, задавая заранее определенные значения полей (значения по умолчанию). Ввод значения в поле может быть заменен выбором нужной позиции из списка возможных значений. Кроме того, при наличии повторяющихся значений полей в следующих друг за другом записях СУБД позволяют переносить соответствующие значения из предыдущей записи в следующую за ней запись. Для сокращения ввода данных можно формировать отдельные поля автоматически, используя для этого функции или иные выражения.

Как отмечалось выше, в большинстве случаев в БД не следует хранить производные показатели. Если же по каким-либо причинам это признано целесообразным, то значения этих полей следует не вводить вручную, а вычислять. Это не только сократит объем ручных операций и возможность ошибок при вводе, но и обеспечит автоматическое поддержание целостности данных.

При организации ввода данных важно не только выявить возможные ошибки, но и организовать ввод таким образом, чтобы предотвратить их.

Одним из приемов, предотвращающим возможные ошибки и упрощающим работу пользователей при вводе данных, является использование

функций по преобразованию данных. Так, например, символьные данные могут храниться в поле базы данных в виде заглавных либо строчных букв. Чтобы пользователь не задумывался, какими именно буквами должны вводиться данные, не тратил усилия на переключение клавиатуры, можно использовать соответствующие функции, преобразующие символы из одного регистра в другой, и шаблоны.

Для обеспечения удобства ввода данных в БД обычно создают специальные экранные формы. При добавлении записи может оказаться полезным использовать два экрана ввода. Первый предназначен только для ввода значения ключа. Он будет использоваться для проверки на наличие дубликатов, прежде чем будет продолжен ввод, и выдачу сообщений в случае ошибок. Второй экран будет использоваться для ввода остальных данных.

Существует несколько вариантов ввода записей:

1. сразу в БД;
2. сначала в переменные памяти (при вводе данных в переменные памяти они могут быть обособленными переменными, а могут быть организованы в виде массива);
3. с использованием вспомогательных файлов.

Второй и третий из перечисленных вариантов позволяют переносить введенные данные в БД только после того, как будут выполнены все необходимые проверки.

Третий вариант дает, кроме того, возможность распараллелить процесс ввода данных.

При корректировке БД могут использоваться те же самые формы, что и при первоначальном вводе, или, что чаще, специально созданные экранные формы. В экранных формах, предназначенных для корректировки, может быть запрещено изменение отдельных полей. Так же как и при вводе данных, экранные формы для корректировки для удобства пользователей лучше связывать не со структурой корректируемого файла, а с функциями, выполняемыми пользователем.

При осуществлении корректировок следует знать ограничения, накладываемые СУБД. Так, некоторые системы не разрешают корректировать ключевое поле. В других системах допускается корректировка ключевых полей и полей индексирования. Но в любом случае корректировка ключа считается плохой практикой. Рекомендуется проектировать БД таким образом, чтобы не было причин для корректировки ключа. Например, если естественный идентификатор объекта является динамическим (т.е. может со временем меняться), то в БД следует ввести искусственный идентификатор и использовать его в качестве ключа. Не следует также повторно использовать идентификаторы удаленных объектов.

Корректировка ключа приводит ко многим нежелательным последствиям. Каскадное обновление, реализованное во многих современных СУБД, иногда не решает всех проблем, связанных с корректировкой ключа.

Особую осторожность нужно проявлять и при корректировке проиндексированных файлов. Здесь надо обращать внимание на то, осу-

ществляется ли корректировка индексного файла одновременно с соответствующим файлом БД системой автоматически. При корректировке поля индексирования ввиду того, что записи в проиндексированных файлах обрабатываются в логической последовательности, а сама эта последовательность меняется в результате корректировки, может случиться, что не все требуемые записи будут обработаны. В связи с этим не рекомендуется делать операции по корректировке поля индексирования при активном индексе, особенно это касается случаев, когда корректируются не единичные записи, а группа записей, отбираемых по условию FOR или WHILE. Даже если корректировка индексных файлов не приведет к некорректностям, нужно избегать поддержания многих индексных файлов открытыми, особенно при массовой корректировке, поскольку это замедляет обработку.

Следует обратить внимание на то, как (в какой момент, какими «порциями» и др.) запоминаются данные в БД. Некоторые СУБД позволяют управлять этим процессом, в других - он жестко предопределен. Как правило, перенос в БД сразу большого количества записей (в противовес позаписному) ускоряет обработку данных, но приводит к другим отрицательным последствиям. В частности, при вводе большого количества записей может оказаться, что в случае какого-то сбоя (например, отключения питания) некоторая часть введенных записей пропадет. Современные СУБД обладают средствами, позволяющими минимизировать отрицательные последствия, возникающие в таких ситуациях.

При организации ввода данных часто используют маски. Их применение не позволяет ввести в поле значения, не вписывающиеся в маску, что позволяет обеспечивать контроль правильности вводимых данных.

С помощью масок можно организовать ввод конфиденциальной информации (если использовать маску типа «пароль», то вместо символов, введенных в поле, будут изображаться звездочки (*); если в СУБД нет специальных типов, предназначенных для подобных целей, то можно запрограммировать эту возможность самостоятельно (например, сделать одинаковым цвет букв и фона).

Введение в маску текстовых констант, во-первых, освобождает от внесения их вручную при вводе данных и, во-вторых, позволяет не хранить их в каждом из полей.

При выполнении корректирующих операций с базой данных СУБД проверяет соблюдение заданных ограничений целостности. При обнаружении ошибочной ситуации могут быть предприняты различные действия, которые зависят от типа допущенной ошибки, специфики предметной области, особенностей СУБД, принятых проектных решений при создании СОД. Так, если допущена ошибка в типе данных или неправильно введена дата, то пользователь должен обязательно исправить ошибку, поскольку СУБД не дает других возможностей продолжить работу. В зависимости от конкретной ситуации могут быть разработаны специальные программы обработки ошибочных ситуаций.

5.3.2. Возможности ввода данных в Access

Ввод данных в таблицу может быть осуществлен в режиме **Таблица** при работе с объектом ТАБЛИЦА.

Если таблица была создана ранее и в настоящий момент она закрыта, то попасть в режим **Таблица** можно, позиционировавшись на вкладке **Таблица** в окне БД на имени нужной таблицы и щелкнув по кнопке **Открыть**.

При создании новой таблицы после описания ее структуры можно сразу перейти в режим **Таблица** и приступить к вводу данных.

Каждая таблица содержит пустую запись, которая следует за последней существующей записью и предназначена для ввода новых данных. Эта запись отмечена слева символом «звездочка» (*). Позиционироваться на эту запись можно разными способами, например, щелкнув по соответствующей кнопке **Новая запись** в панели перемещения по записям или выбрав позицию меню **Вставка/Новая запись**, или просто мышью.

Введенные данные автоматически сохраняются при переходе к другой записи.

Для ускорения ввода данных в текущее поле таблицы могут быть использованы определенные комбинации клавиш:

Клавиша	Действие
[Ctrl]+[;]	Вводит текущую дату
[Ctrl]+[:]	Вводит текущее время
[Ctrl]+[Alt]+[Пробел]	Вводит значение поля, установленное по умолчанию
[Ctrl]+['](апостроф) или ["] (кавычки)	Вводит значение того же поля из предыдущей записи

Для перехода между записями можно использовать кнопки внизу окна таблицы в панели перемещения по записям (к первой, следующей, предыдущей, последней записи).

Для облегчения процесса ввода данных в БД можно использовать соответствующие свойства полей: значение по умолчанию, маска ввода, формат поля.

Значение по умолчанию. Обычно в этом качестве указывается какое-то постоянное значение, однако можно использовать и выражение. Например, для ввода текущей даты можно в качестве значения по умолчанию использовать функцию =Date(). Значение, введенное по умолчанию, может быть впоследствии изменено.

Выражения, которые используются в качестве значений по умолчанию, не должны содержать ссылок на элементы управления и другие поля, а также функций, определенных пользователем. Выражения могут записываться непосредственно или строиться с помощью «Построителя выражений».

Маска ввода (InputMask). Использование маски позволяет определить значения, которые можно ввести в поле, и не только обеспечивает контроль правильности ввода данных, но и облегчает их ввод. Значение свойства «Маска ввода» может быть введено вручную или с помощью Мастера масок ввода.

Используемые в масках знаки и их интерпретация приведены в приложении 1.

Рассмотрим некоторые примеры использования масок ввода.

1. Маска ввода для поля «Телефоны» может выглядеть следующим образом:

\(999")-"999\-9999

В приведенном примере константы (скобки, дефисы) отражаются на экране, но в базу данных не вводятся.

2. Пусть на фирме адрес электронный почты (ЭП) сотрудников формируется следующим образом: префикс - любые четыре символа, а затем для всех указывается @firm.ru

В этом случае маска может выглядеть следующим образом:

AAAA"@ firm "."ru";0

Значение свойства «Маска ввода» может содержать до трех разделов, разделенных точкой с запятой (;). Первый представляет саму маску ввода, второй - определяет режим занесения в таблицу текстовых констант, добавляемых к знакам, вводящимся пользователем. «О» в данном компоненте указывает, что текстовые константы сохраняются вместе с введенными пользователем значениями; значение «1» или пустое значение данного раздела указывает, что сохраняются только знаки, введенные пользователем. В данном примере постоянная часть адреса электронный почты сохраняется в БД. Третий раздел определяет знак, используемый для изображения пустых позиций в маске ввода, в которые помещаются вводящиеся пользователем знаки.

Свойство «Формат поля» (Format). Кроме свойства «Маска ввода» в Microsoft Access можно задать еще свойство «Формат поля» (Format). Их использование приводит к похожим результатам.

Свойство «Формат поля» влияет только на отображение значения, но никак не влияет на значение, хранимое в таблице. Для управления вводом данных удобнее использовать маску ввода.

Если для поля определены как формат отображения, так и маска ввода, то при добавлении и редактировании данных используется маска ввода, а формат поля определяет отображение данных после сохранения записи.

Несмотря на то, что последние версии Access позволяют в режиме таблицы вводить данные даже во взаимосвязанные таблицы, для организации ввода данных в базу данных в большинстве случаев следует использовать экранные формы.

На это следует обратить внимание

- Организуя ввод данных в БД, помните, что человек является самым ненадежным и самым дорогим элементом информационной системы.
- При организации ведения баз данных нужно стремиться к реализации принципа однократного ввода информации.
- Старайтесь до минимума сократить количество ручных операций.
- Обеспечивайте контроль правильности введенных данных.

Контрольные вопросы

1. Что в Access называется базой данных?
2. К какому классу относится СУБД Access?
3. Каковы особенности реляционной модели данных?
4. Как создать новую базу данных в Access?
5. Как добавить новый объект в существующую базу данных?
6. Какие способы создания таблиц вы знаете? В каких случаях следует использовать каждый из них?
7. Какие типы полей допустимы в Access? Каковы особенности работы с полями каждого из этих типов?
8. Какие способы создания полей подстановки вы знаете? В каких случаях следует использовать каждый из них?
9. Какие преимущества дает использование полей подстановки?
10. Какие ограничения накладываются на имена полей?
11. Что называется ключом таблицы? Какие разновидности ключей вы знаете?
12. Какими способами можно создать ключ?
13. Является ли наличие ключа в таблице Access обязательным?
14. В каких случаях задание ключа является обязательным?
15. Какими специфическими особенностями обладает поле типа «Счетчик»?
16. Какие свойства полей вы знаете? Приведите примеры их использования.
17. Как можно изменить структуру существующей таблицы?
18. Как можно задать объединение таблиц? Какие способы объединения вы знаете? Как можно изменить тип объединения?
19. Какие способы задания ограничений целостности в Access вы знаете?
20. Как задается в Access «ограничение целостности связи»?
21. Какие способы ввода данных в БД вы знаете? Назовите достоинства, недостатки и сферы применения каждого из этих способов.

Глава 6 ЯЗЫК ЗАПРОСОВ QBE

6.1. Общая характеристика языка QBE

В современных СУБД широко используются табличные языки запросов. Наиболее распространенным среди них является язык QBE (Query-By-Example - запрос по примеру). Язык QBE предназначен для работы в интерактивном режиме и ориентирован на конечного пользователя. Язык QBE реализован во многих современных СУБД, например в dBase IV и более старших версиях этой системы, Paradox, Access и др. Конкретные реализации этого языка несколько отличаются друг от друга, но все они построены по единому принципу.

Суть подхода, воплощенного в языке QBE, заключается в следующем. В окне формирования запроса выделяются две зоны. В первой из них высвечивается «скелет» (образ, форма, структура) одной или нескольких таблиц, данные из которых будут участвовать в запросе. В качестве исходных для запроса могут указываться не только базовые таблицы, но и другие запросы.

Во второй зоне («скелете» запроса табличной формы) пользователь задает условия запроса. В этой зоне пользователь определяет, какие поля участвуют в формировании запроса, а также условия отбора и некоторые другие характеристики запроса. Например, если пользователю необходимо получить все записи с заданным значением конкретного атрибута, то в соответствующем столбце «скелета» указывается это значение.

На рис. 6.1 представлен запрос к таблице, содержащей сведения о сотрудниках (Kadr) и включающей следующие атрибуты:

- FAM - фамилия;
- IMIA - имя;
- TABN - табельный номер;
- VOZR - возраст;
- POL - пол;
- ADR - адрес.

Kadr	FAM	IMIA	TABN	VOZR	POL	ADR
				40		

Рис. 6.1. Простой запрос

Требуется выдать информацию обо всех сотрудниках в возрасте 40 лет. В соответствующем столбце таблицы (VOZR) указывается цифра 40. В столбце можно записывать не только значение атрибута, но и знак операции сравнения; по умолчанию принимается знак равенства («=»).

Задание сложных запросов. Допускается задание и простых запросов, включающих только один аргумент поиска, и сложных запросов, компонен-

ты которых связаны операторами AND (И) или OR (ИЛИ). Операторы AND и OR в явном виде не указываются при формулировании запроса на QBE. При отображении запросов на экране используется следующее правило: если в сложном запросе его компоненты представляют разные атрибуты, которые должны быть связаны оператором AND, то они записываются в одной строке (рис. 6.2). Если компоненты запроса должны быть связаны операторами OR, то они записываются на разных строках (рис. 6.3).

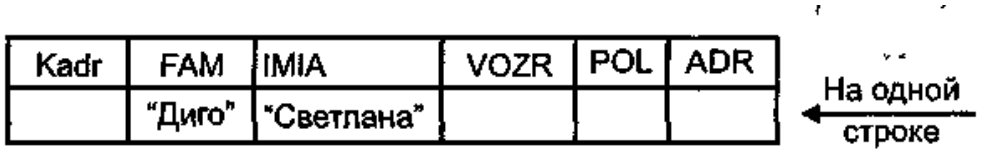


Рис. 6.2. Сложный запрос с оператором AND

На рис. 6.2 изображен запрос: «Выдать информацию о сотруднике с фамилией Диго и именем Светлана», а на рис. 6.3 - «Выдать информацию о сотрудниках, имеющих либо фамилию Диго, либо имя Светлана».

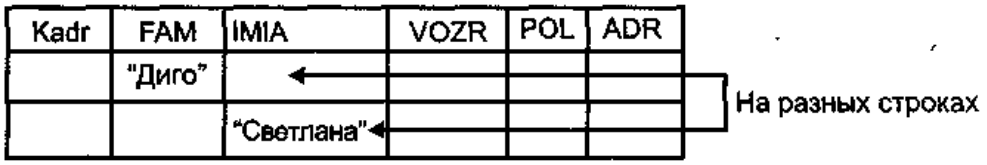


Рис. 6.3. Сложный запрос с оператором OR

В связи с тем, что интерпретация запроса зависит от взаимного расположения элементов сложного запроса на строках экрана, такого рода языки запросов называются *табличными двухмерными*.

Как указывалось выше, при задании запроса в QBE экран обычно делится на две зоны: зона, в которой указываются данные, исходные для запроса, и зона, в которой описывается ответ. В некоторых реализациях языка при описании отдельных видов запросов появляются дополнительные зоны (например, в dBase IV при задании вычисляемого поля [19]).

Вид, в котором представляются структуры исходных таблиц, а также то, где фиксируются условия поиска, могут различаться в конкретных системах.

Так, в dBase IV таблицы как в зоне «запроса», так в зоне «ответа» представляются в табличном виде, а условия отбора записей указываются в таблицах зоны «запроса». В Access, FoxPro исходные таблицы представлены в анкетной форме (поля таблицы перечисляются один под другим), а в зоне «ответа» в табличной форме отображаются те атрибуты (поля), которые будут выдаваться в ответе. Условия отбора записей задаются в зоне «ответа».

Переменные для примера. В некоторых случаях при формулировке запроса необходимо использовать так называемые переменные для примера (или «наполнители»). Переменные для примера (example variables) также записываются в определенных графах таблицы, но они обозначают не какое-

либо определенное значение, а любое. Конкретное значение наполнителя несущественно. Переменные для примера используются для установления связей между атрибутами в одной или нескольких таблицах.

Переменные, применяемые для задания значений ключей поиска, и переменные, указываемые для примера, должны при записи запроса отличаться друг от друга.

В разных СУБД «наполнители» и обычные значения атрибутов поиска различаются по-разному: в некоторых системах «наполнители» подчеркиваются, в других - используются специальные ограничители при указании переменных в запросе, в третьих - такое понятие вообще не вводится и т.п.

Совместная обработка нескольких таблиц. В некоторых запросах могут потребоваться данные из нескольких таблиц. Например, в базе данных, кроме таблицы KADR, имеется таблица «Выработка» (VRBT) с полями:

- TABN - табельный номер;
- DAT - дата;
- KODDET - код детали;
- KOLV - количество.

В запросе «Выдать информацию о выработке рабочего Евгения Петрова» необходима совместная обработка таблиц VRBT и KADR, так как в таблице «Выработка» нет сведений о фамилиях и именах рабочих.

«Скелеты» всех таблиц, которые нужны для реализации запроса (в нашем примере - двух таблиц), должны быть вызваны на экран.

Дальнейшие действия, которые необходимо выполнить, чтобы осуществить связывание таблиц, будут зависеть от используемой СУБД. Так, в некоторых системах для связывания таблиц используются «наполнители». Их значения могут быть любыми, но они должны быть одинаковыми в обеих связываемых таблицах.

В примере, представленном на рис. 6.4, в качестве наполнителя используется буква А, и она подчеркивается.

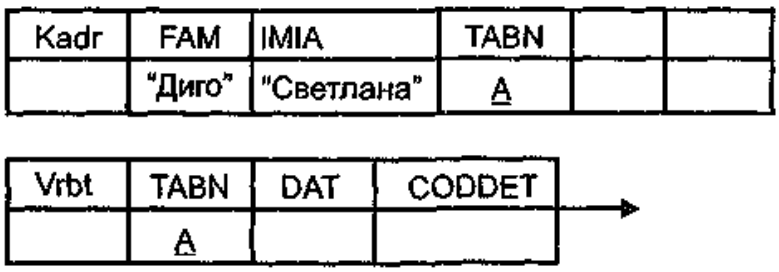


Рис. 6.4. Запрос со связыванием таблиц

В более поздних версиях СУБД используются визуальные способы установления связей между таблицами: для связывания таблиц нужно мышью позиционироваться на нужном поле в основной таблице и, не отпуская кнопки мыши, переместиться к полю в зависимой таблице. На экране появится линия, связывающая таблицы.

Существуют и другие способы установления связей.

Теоретически возможны разные типы соединений таблиц. Наиболее распространенным является соединение, при котором в результатную таблицу помещаются те соединенные записи, для которых значение поля связи основной таблицы совпадает с соответствующим полем в зависимой таблице. В описанных выше случаях устанавливается именно такое соединение. В настоящее время широко используются такие понятия, как «левое» и «правое» соединение, когда в результатную таблицу помещаются все записи из основной или зависимой таблицы соответственно, даже если для них нет связанных записей в другой таблице. Но не все системы позволяют в QBE реализовывать такие соединения. В случаях, когда возможно задание разных типов соединений, конкретный способ реализации отличается в разных СУБД. Так, в Access «левое» и «правое» соединения можно определить, задав для связи «параметры объединения» или перейдя в SQL. В dBase IV никаких специфических терминов для обозначения такого типа соединений нет, но включение слова Every в запрос на QBE выполняет ту же роль.

Работа с несколькими таблицами в конкретных СУБД различается не только тем, каким способом можно определить связь между таблицами. Так, например, некоторые системы обязывают пользователя связать те таблицы/файлы, которые указываются как исходные для запроса; другие автоматически связывают открытые файлы по тем полям, которые система воспринимает как поля связи (чаще всего это поля, имеющие одинаковые имена, тип и длину); третьи - оставляют эти таблицы изолированными, если пользователь не указал, как они должны быть связаны, четвертые - выполняют декартово произведение открытых таблиц. Например, в dBase IV вызвать несколько файлов БД на панель запросов и не связать их было нельзя. В MS Query, Access если таблицы не связаны, то при выполнении запроса это приводит к связыванию каждой записи одной таблицы с каждой записью другой.

> **Внимание!** Будьте внимательными при реализации запросов, в которых открыты несколько таблиц. Не открывайте таблиц больше, чем это действительно требуется для реализации каждого конкретного запроса!

Описание ответа. Кроме задания условия отбора данных, при описании запроса должна быть возможность указать, какие атрибуты и в какой последовательности входят в ответ. В ответ могут выдаваться не только реальные поля, которые хранятся в одной из базовых таблиц, но и вычисляемые поля.

Можно выделить два вида вычислений, которые могут выполняться в запросах, формах, отчетах: это агрегирующие операторы, которые выполняют операции над группой записей, и обычные вычисления, затрагивающие отдельные поля одной или нескольких связанных записей.

Агрегирующие показатели могут быть включены не только в «Запросы», но и в «Отчеты». Возможности включения агрегирующих показателей в запросы и отчеты различаются между собой. Результатом запроса всегда является плоская таблица. Поэтому в запросах могут быть получены только од-

ноуровневые итоги. В отчетах же может быть получено несколько степеней итогов.

Набор агрегирующих функций может быть различным в разных системах. Обычно во всех реализациях СУБД включены следующие функции: Sum (сумма), Min (минимум), Max (максимум), Avg (среднее), Count (подсчет). Некоторые системы включают дополнительные статистические функции, такие, как отклонение, стандартное отклонение, дисперсия и др.

Использование агрегирующих функций предполагает, что таблица упорядочена по тому полю (полям), по которому ведется агрегирование. Некоторые СУБД сами автоматически выполняют упорядочение данных по необходимым полям, другие - нет. В последнем случае, если пользователь не задаст правильно требуемое упорядочение, результат, выводимый в ответ, будет искаженным.

Результаты вычислений, выводящиеся в поле, не запоминаются в базовой таблице. Вместо этого вычисления снова проводятся всякий раз, когда выполняется запрос, поэтому результаты всегда представляют текущее содержимое базы данных. Обновить вычисленные результаты вручную невозможно (таблица, содержащая вычисляемое поле, имеет статус «только для чтения»).

Для удобства восприятия ответа часто требуется определить упорядоченность данных в ответе. Язык QBE обеспечивает такую возможность. Опять-таки возможности задания упорядочения ответа различаются в разных СУБД: некоторые системы разрешают проводить упорядочение по произвольным полям, другие требуют, чтобы поле упорядочения стояло в ответе обязательно первым, а если упорядочение ведется по нескольким полям, то чтобы эти поля следовали в ответе друг за другом в порядке их старшинства; некоторые СУБД различают обычное и словарное упорядочение (когда учитывается и не учитывается регистр соответственно), другие - нет; в некоторых системах, даже если не задано никакое упорядочение, ответ всегда выдается упорядоченным по первому полю таблицы ответа и т.п.

Дополнительные возможности. Кроме собственно поисковых запросов язык QBE позволяет выполнять и другие операции, например корректировку данных. Набор допустимых операций, а также способы их задания несколько различаются в разных системах.

Кроме того, некоторые СУБД позволяют формировать запросы специальных видов: параметрические, перекрестные и некоторые другие (не все из них, наверное, могут быть отнесены к QBE, но они реализованы одними и теми же компонентами СУБД и в связи с этим будут здесь рассмотрены).

Запросы, сформулированные на QBE, могут быть запомнены для их последующего многократного использования.

Некоторые языки запросов, которые носят название QBE, построены совсем на других принципах, чем те, что были изложены выше, и было бы хорошо найти для них другое название. Так, например, язык RQBE FoxPro не является табличным двухмерным языком запросов. Он является «построителем» запросов (в том числе SQL). Сложный запрос реализуется в нем просто

вводом каждого элементарного условия на отдельной строке. Если две строки не разделены никакой операцией, то считается, что они соединяются операцией «И». Операцию «ИЛИ» нужно указывать явно между соединяемыми строками.

Все языки запросов имеют много общего. Но, с другой стороны, даже языки, относящиеся к одному классу, отличаются в деталях своей реализации. Начиная работать с новой для вас СУБД, проведите серию экспериментов, чтобы убедиться, что ваше понимание логики реализации запросов соответствует той, что заложена в данной конкретной системе.

6.2. Реализация QBE в Access

6.2.1. Общие сведения

После описания таблиц и заполнения их данными к базе данных можно формулировать разнообразные запросы. В табличном языке запросов в Access реализованы те общие принципы, о которых шла речь выше. Но, естественно, имеются и особенности реализации языка в этой конкретной системе.

В явном виде понятие наполнителя в QBE в Access не вводится. Связывание таблиц выполняется визуально. Для задания в запросе соотношения полей внутри таблицы вместо наполнителя можно просто задать соответствующее выражение для вычисляемого поля и для этого выражения задать условия отбора.

Для задания запроса в Access следует перейти к объекту **Запросы** в окне базы данных. Для создания нового запроса следует щелкнуть по кнопке **Создать** - появится окно **Новый запрос** (рис. 6.5).

Для построения запросов могут быть использованы различные мастера. Но чаще всего используется возможность создания запроса с помощью **Конструктора**.

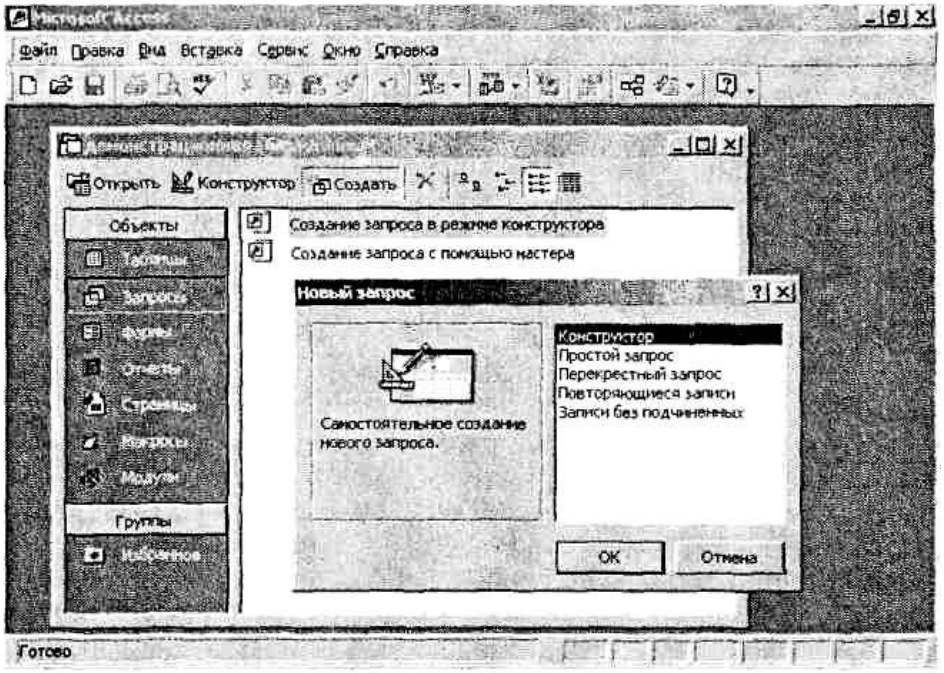


Рис. 6.5. Вид окна **Новый запрос**

Первым шагом при создании запроса является определение таблиц, которые содержат исходную информацию. Допускается также создание запроса на основании других запросов или одновременно и таблиц, и запросов. Использование предварительно созданных запросов при создании нового запроса может помочь сделать сложный запрос, содержащий большое число взаимосвязанных таблиц и много разнообразных условий отбора, более простым для его формулирования.

Любой запрос можно сохранить для последующего использования. Это можно сделать несколькими способами, например, выбирая позиции меню **Файл/Закрыть**, ответив «Да» на вопрос о сохранении файла и задав после этого имя запроса. Имя запроса не может совпадать не только с именем другого запроса, но и с именем какой-либо таблицы в данной базе данных.

Сохраненный запрос можно впоследствии «открывать», что означает его выполнение. Сохраненный запрос может быть скорректирован, если открыть его в режиме **Конструктор**.

Добавление таблиц в запросе

Возможно использование нескольких способов включения таблиц, на которых базируется запрос.

Для добавления таблицы можно в режиме **Конструктор запроса** щелкнуть по кнопке **Добавить таблицу** на панели инструментов или выбрать в меню **Запрос** команду **Добавить таблицу** - открывается окно диалога **Добавление таблицы.**

Пользователь имеет также возможность добавить в запрос таблицу или запрос, выбрав их имена в окне базы данных и переместив их с помощью мыши в верхнюю часть окна запроса.

Кроме того, можно позиционироваться на свободное место в верхней части окна запроса, нажать правую кнопку мыши и в появившемся ниспадающем меню выбрать позицию **Добавить таблицу**.

Для добавления в запрос таблицы из другой базы данных или другого приложения следует сначала присоединить эту таблицу к активной базе данных.

Это присоединение выполняется путем использования команды **Присоединить таблицу** (меню **Файл**).

При создании нового запроса без помощи мастера по разработке запросов окно диалога **Добавление таблицы** выводится автоматически (рис. 6.6). Для того чтобы указать, на чем будет базироваться создаваемый запрос (таблице, запросе или том и другом одновременно), нужно просто выбрать соответствующую вкладку.

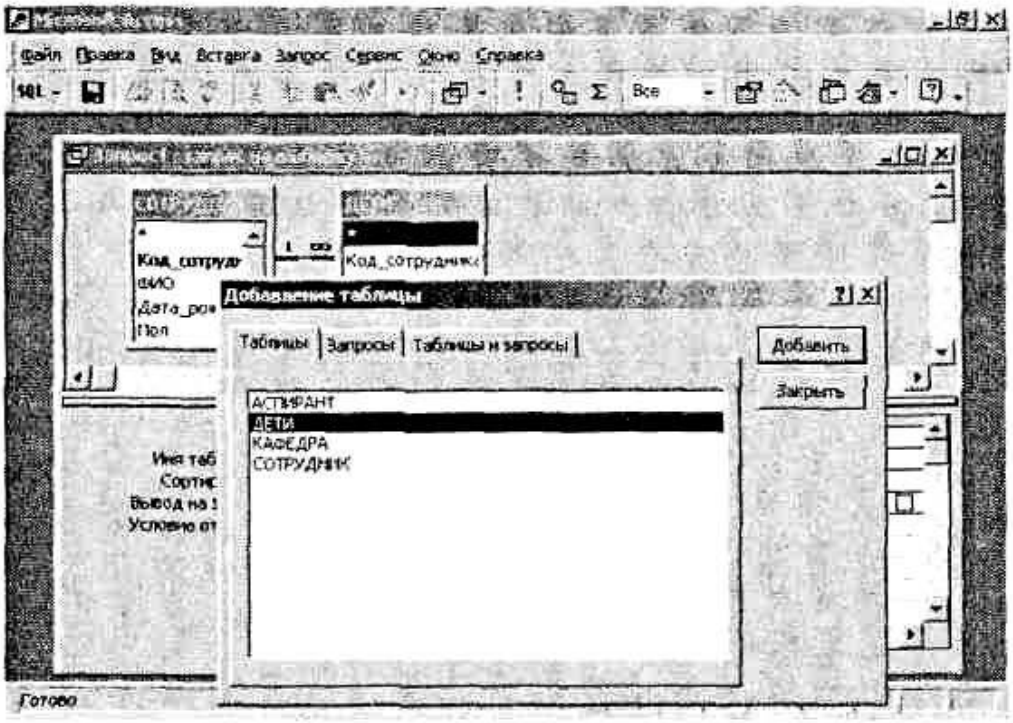


Рис. 6.6. Выбор источника запроса

Установив в появившемся списке доступных таблиц/запросов указатель на имя добавляемой таблицы или запроса, нужно или выполнить двойной щелчок «мышью», или нажать клавишу [Enter].

Допускается одновременное добавление в запрос нескольких таблиц или запросов. Для этого следует, удерживая нажатой клавишу [Ctrl], выбрать имена добавляемых таблиц или запросов и щелкнуть по кнопке **Добавить**.

В верхней части окна запроса выводится список полей добавленных таблицы или запроса.

Действия, описанные выше, повторяются для добавления других таблиц или запросов.

Удаление таблицы из запроса

Существует несколько способов удаления таблицы из запроса:

- выбрать имя удаляемой таблицы или запроса в соответствующем списке и нажать клавишу [Del] или выбрать в меню **Запрос** команду **Удалить таблицу**;
- двойным щелчком мыши выделить нужную таблицу в соответствующей зоне экрана и затем нажать клавишу [Del].

Имена полей удаленной таблицы или запроса удаляются из бланка запроса QBE. Удаление из запроса таблицы или запроса, на которых он базируется, не приводит к их удалению из базы данных.

Различают несколько типов запросов: запрос на выборку (Select), перекрестный запрос (Crosstab), создание таблицы (Make-table), запрос на обновление (Update), добавление (Append), удаление (Delete). Наиболее часто используемым типом запросов является запрос на выборку. Именно с него мы и начнем изучение возможностей задания запросов в Access.

6.2.4. Включение полей в запрос

После того как определены исходные таблицы/запросы, нужно выбрать поля, используемые в создаваемом запросе. Существует несколько способов переноса поля в бланк запроса: «буксировка» с помощью мыши, двойной щелчок мыши по имени соответствующего поля в списке полей, выбор поля в раскрывающемся списке полей, который появляется, если нажать на знак «стрелки» в строке Поле бланка запроса.

Можно переносить в бланк запроса не по одному полю, а сразу требуемую совокупность полей. Выделение полей, подлежащих переносу, осуществляется стандартным для Windows-систем способом (нажать клавишу [Shift] для выделения полей, расположенных друг за другом, и, удерживая ее, выбрать первое и последнее поле набора; выбирать поля при нажатой клавише [Ctrl] для полей, располагающихся в произвольном порядке не подряд друг за другом).

Можно перенести в бланк запроса одновременно все поля. Для этого следует установить указатель на заголовок списка полей и дважды щелкнуть кнопкой мыши или установить указатель на символ звездочки (*) и нажать кнопку мыши.

6.2.5. Поля, выводимые в ответ

Не все поля, которые используются при формировании запроса, должны обязательно отражаться в ответе. Так, например, поле может быть необходимо для задания условия отбора, но надобность в его появлении в ответе отсутствует.

Поля, выводимые в ответ, указываются в строке конструктора запроса ***Вывод на экран*** (Show). В соответствующих колонках этой строки указывается знак вхождения поля в ответ («V» - «галочка»).

Есть разница, как поля были введены в запрос. При использовании символа звездочки в запрос автоматически включаются все поля, добавленные в базовую таблицу/запрос после создания данного запроса. Все удаленные из структуры таблицы поля будут автоматически удаляться из запроса. С одной стороны, это хорошо, с другой - может случиться, что пользователь в ответ на один и тот же запрос будет получать разный ответ, и, вполне возможно, не тот, который он ожидает. Так, например, если в таблице «Сотрудник» первоначально фиксировались только основные данные по сотруднику, а затем было введено много других полей, то совсем не обязательно, что пользователь захочет видеть все эти данные в ответ на свой запрос.

Если же поле, включенное в запрос явным способом, было впоследствии удалено из таблицы, то запрос может выполняться не совсем корректно.

Поскольку поля, включенные в запрос путем использования «*», в явном виде в бланке запроса не высвечиваются, то те поля, которые используются в условии отбора, нужно дополнительно включить в бланк запроса. Чтобы эти поля дважды не выводились в ответ, следует у этих полей снять флажок **Вывод на экран** (рис. 6.7).

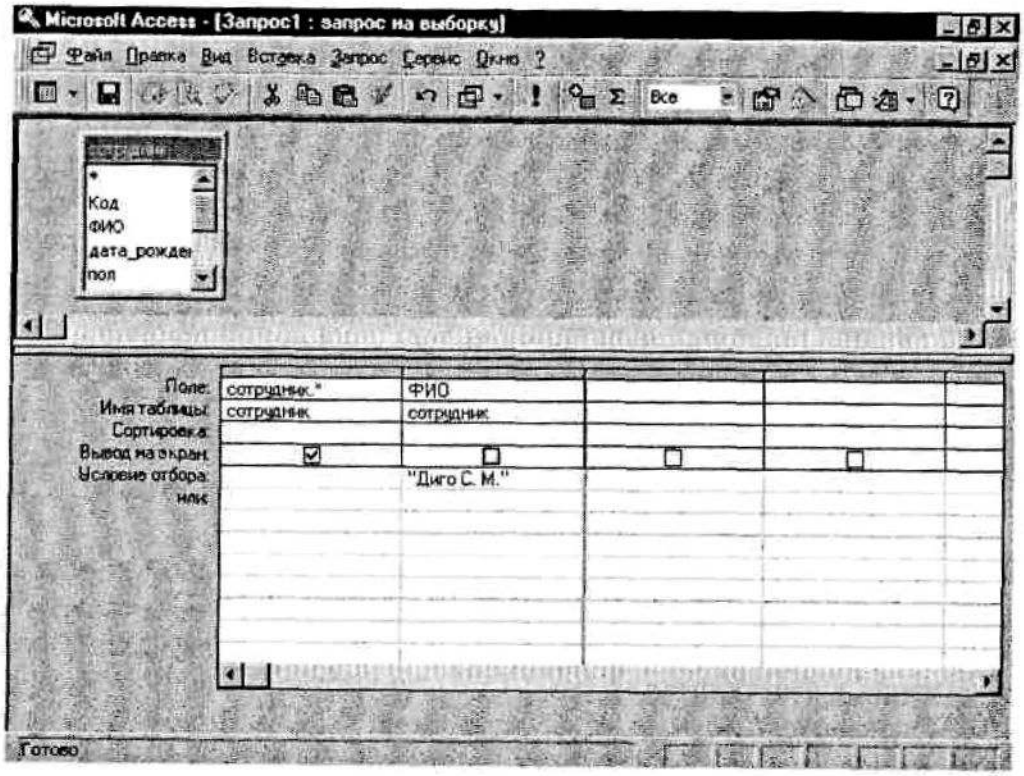


Рис. 6.7. Использование «*» в запросе

6.2.6. Управление выводом повторяющихся строк

В том случае, если в ответ выводятся не все поля исходной таблицы, может случиться, что строки в ответе могут быть повторяющимися. Для того чтобы управлять выводом повторяющихся строк, можно позиционироваться на произвольное место вне бланка запроса и списка полей, нажать правую кнопку мыши и в появившемся контекстном меню (рис. 6.8) выбрать строку «Свойства» (либо выбрать соответствующую кнопку на панели инструментов). Среди свойств запроса (рис. 6.9) есть два: «Уникальные записи» (UniqueRecords) и «Уникальные значения» (UniqueValues), которые служат указанным целям. Если вы хотите, чтобы в ответ выдавался список кафедр без повторов, задайте для свойства «Уникальные значения» значение «Да».

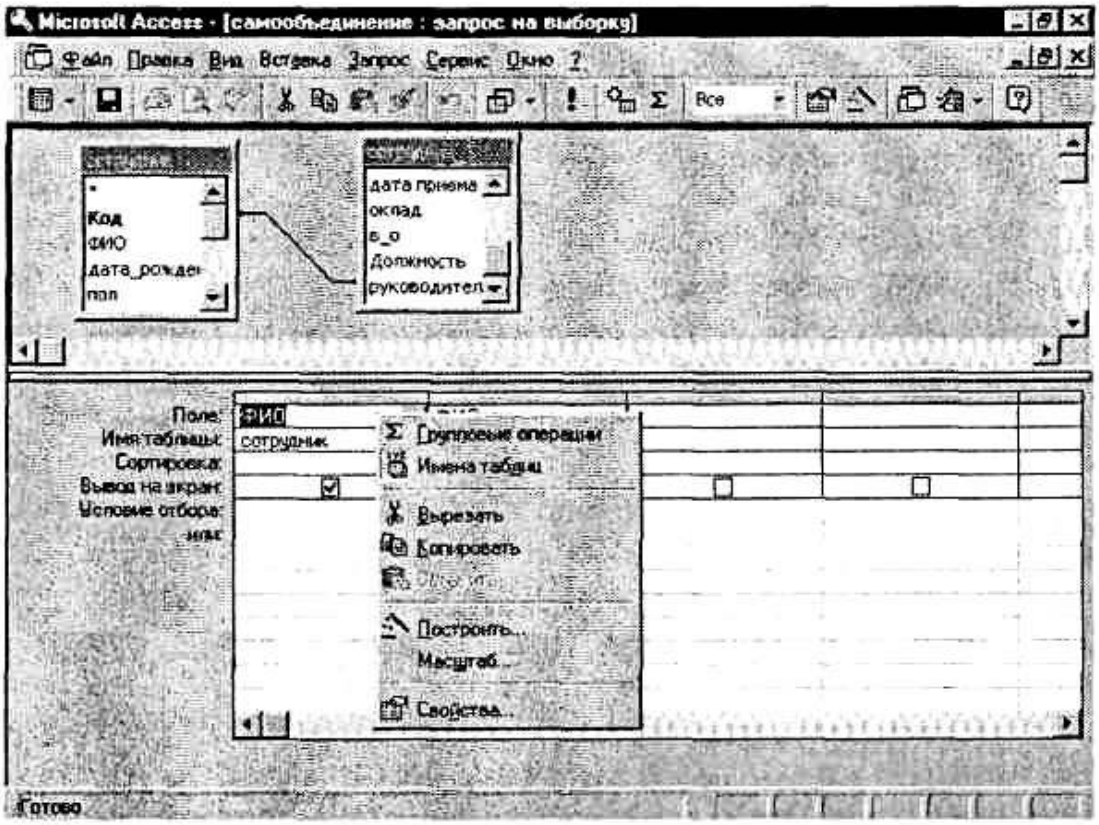


Рис. 6.8. Контекстное меню поля в запросе

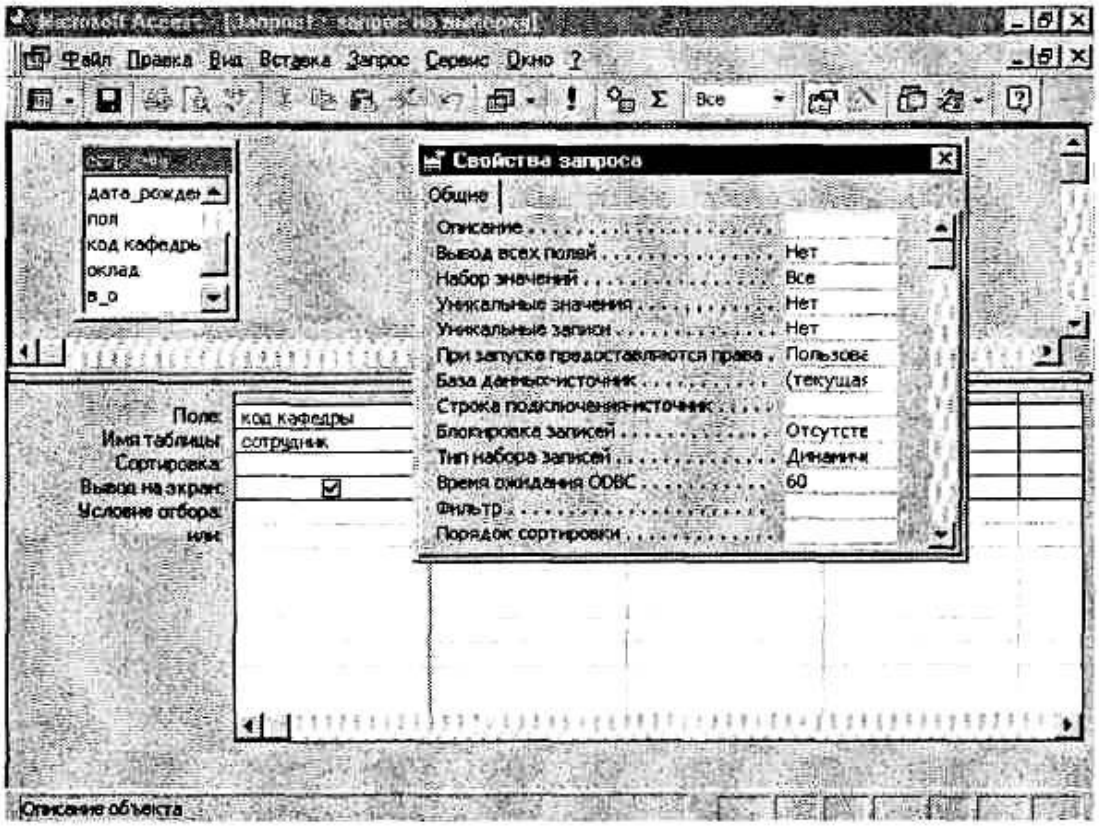


Рис. 6.9. Вид окна **Свойства запроса**

6.2.7. Простые запросы

Запрос с простыми условиями, включающими только один аргумент поиска, будем коротко называть *простым запросом*. При создании простого запроса условие отбора записывается в соответствующий столбец бланка запроса. Например, если требуется отобрать информацию о конкретном сотруднике, то в столбец «ФИО» в строке «Условие отбора» нужно записать ФИО данного сотрудника. В частности, запрос, изображенный на рис. 6.7, является таким простым запросом.

Как известно, в большинстве СУБД при вводе в выражение значений того или иного типа используются соответствующие данному типу данных ограничители. В Access при задании запроса ограничители можно не ставить. В зависимости от типа поля, которое вводится в выражение, определяющее условие отбора, ограничители добавляются системой автоматически: прямые кавычки (" ") вокруг строковых значений; символы (#) вокруг дат.

В столбце можно записывать не только значение атрибута, но и знак операции сравнения; по умолчанию принимается знак равенства (=). Если в условии отбора должны использоваться операции сравнения, отличные от знака равенства, то их надо указывать в явном виде. Если, например, требуется определить список всех сотрудников, имеющих оклад меньше 1000 руб., то запрос будет выглядеть так, как изображено на рис. 6.10.

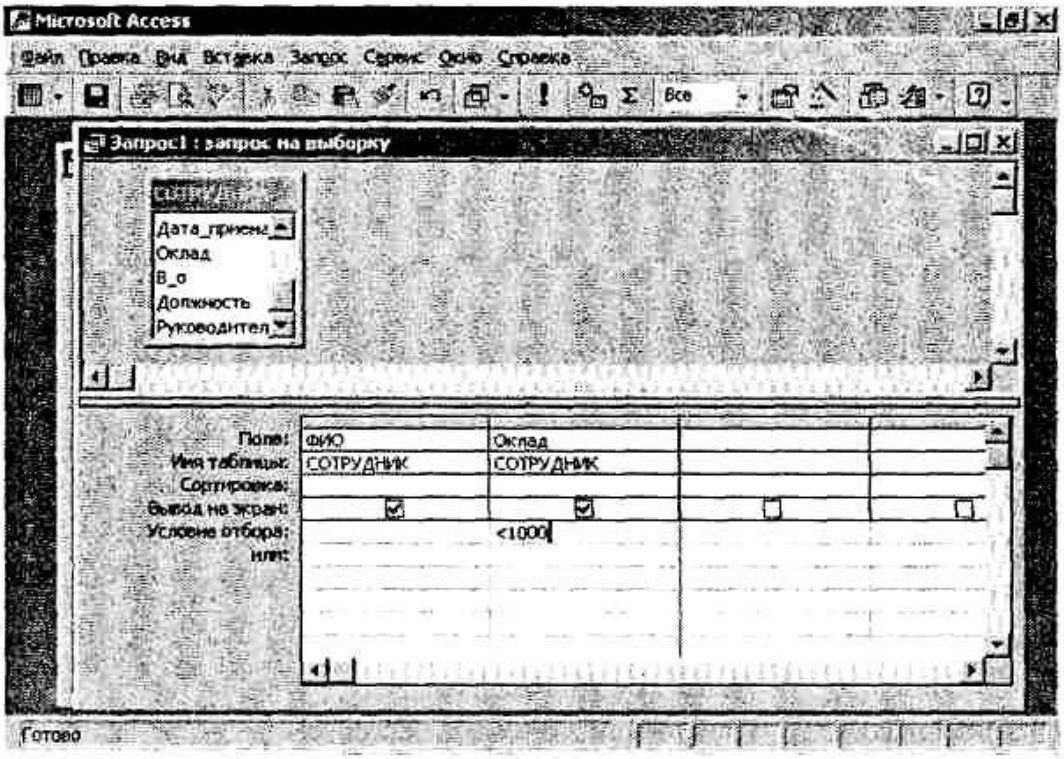


Рис. 6.10. Использование операторов сравнения при задании запроса

В условиях отбора можно задавать и диапазон значений. В этом случае запрос будет выглядеть подобно тому, как изображено на рис. 6.11.

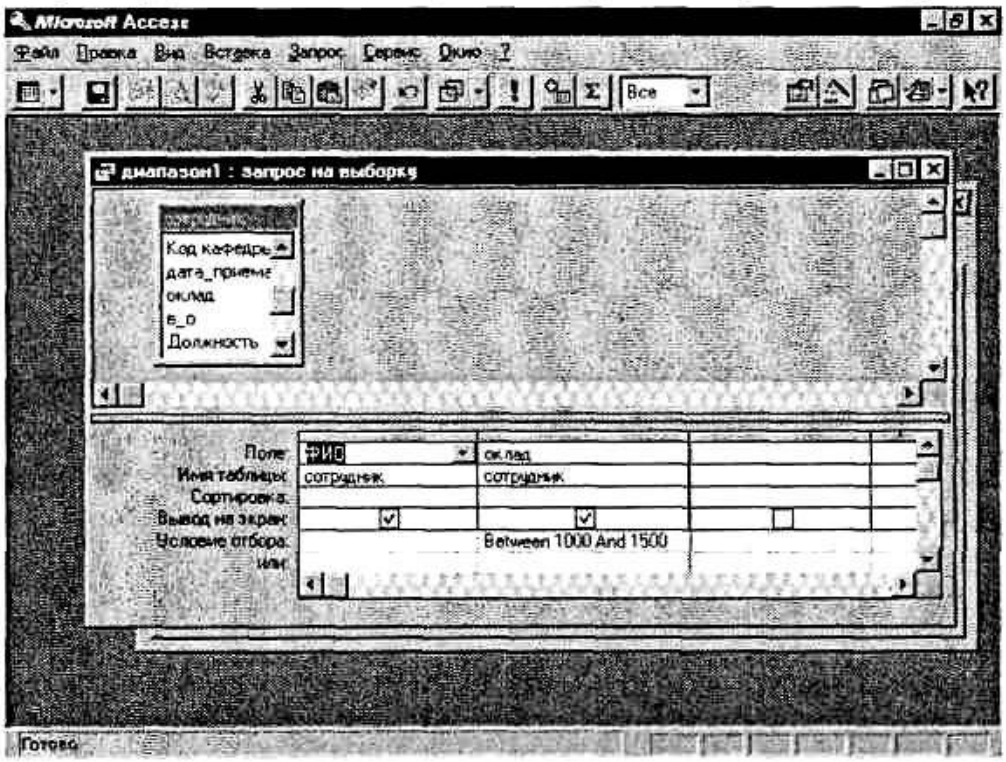


Рис. 6.11. Задание диапазона

Это же условие отбора можно было задать и следующим образом:

>=1000 And>=1500.

Можно осуществлять поиск и по подстроке. Для этого используется оператор Like (рис. 6.12)

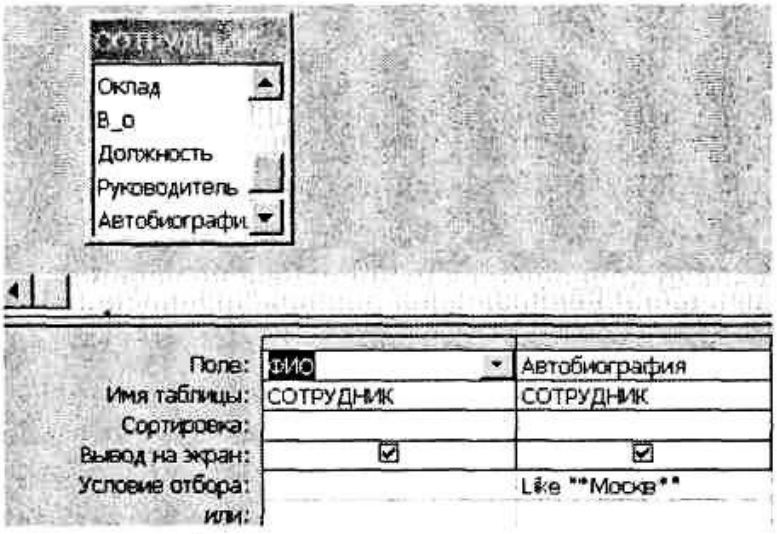


Рис. 6.12. Поиск по подстроке

6.2.8. Сложные запросы

Если в условиях отбора используется несколько полей, то они могут соединяться оператором «И» или «ИЛИ». На рис. 6.13, 6.14 изображены примеры таких запросов. Первый из них выдает список военнообязанных мужчин (запрос «И»; аргументы запроса расположены на одной строке), второй (запрос «ИЛИ»; аргументы запроса расположены на разных строках) - всех мужчин и военнообязанных женщин.

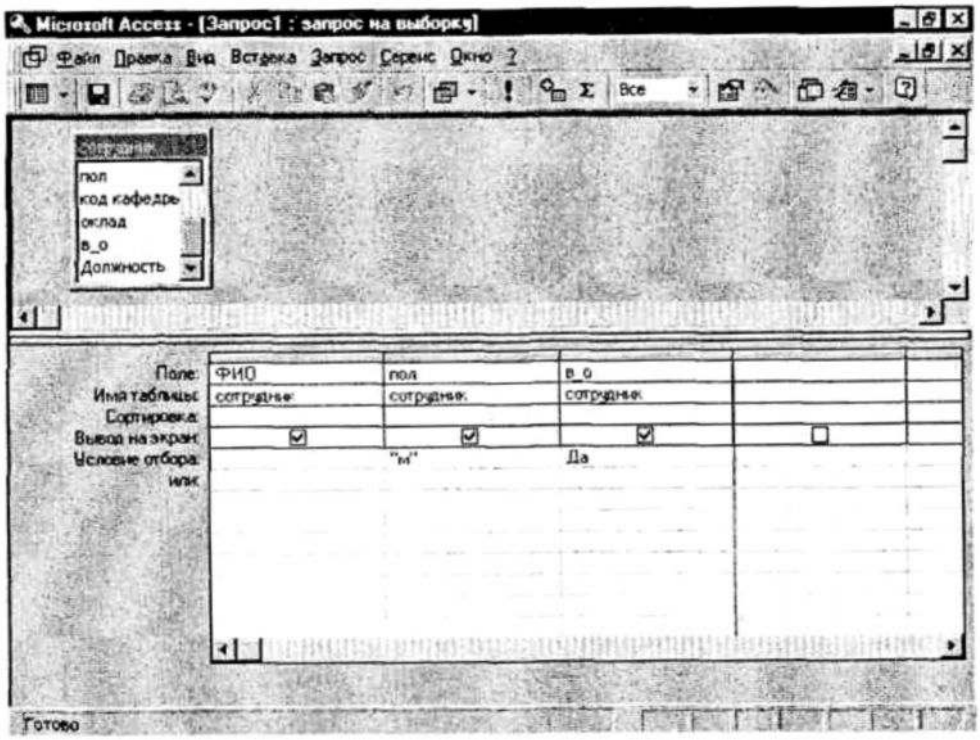


Рис. 6.13. Сложный запрос (оператор AND)

Как видим, разница в примерах запросов, изображенных на рис. 6.13 и 6.14, состоит только в том, что условия отбора заданы в первом случае на одной строке, а во втором - на разных, а ответ при этом получается разный. Поэтому рассматриваемый язык запросов и называется табличным двухмерным языком: ответ зависит от взаимного расположения аргументов поиска относительно друг друга.

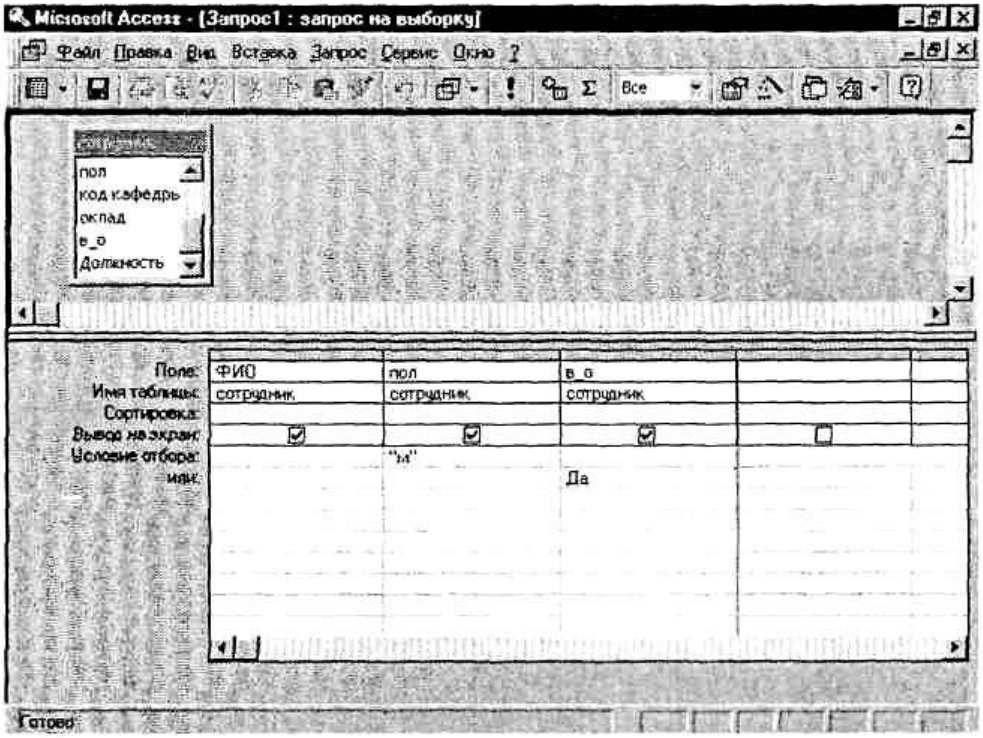


Рис. 6.14. Сложный запрос (оператор OR)

В одном запросе могут использоваться и более двух аргументов поиска, причем одна часть из них может связываться оператором «И», а другая - оператором «ИЛИ».

6.2.9. Просмотр ответа

Для того чтобы посмотреть ответ, можно щелкнуть мышью по кнопке **Запуск** («!») на панели инструментов, либо выбрать соответствующую возможность из меню **Запрос/Запуск**, либо щелкнуть по стрелке на кнопке **Вид** и выбрать из появившегося списка вид **Режим таблицы**. Для того чтобы опять вернуться к построению / корректировке запроса, надо выбрать режим **Конструктор** (рис. 6.15).

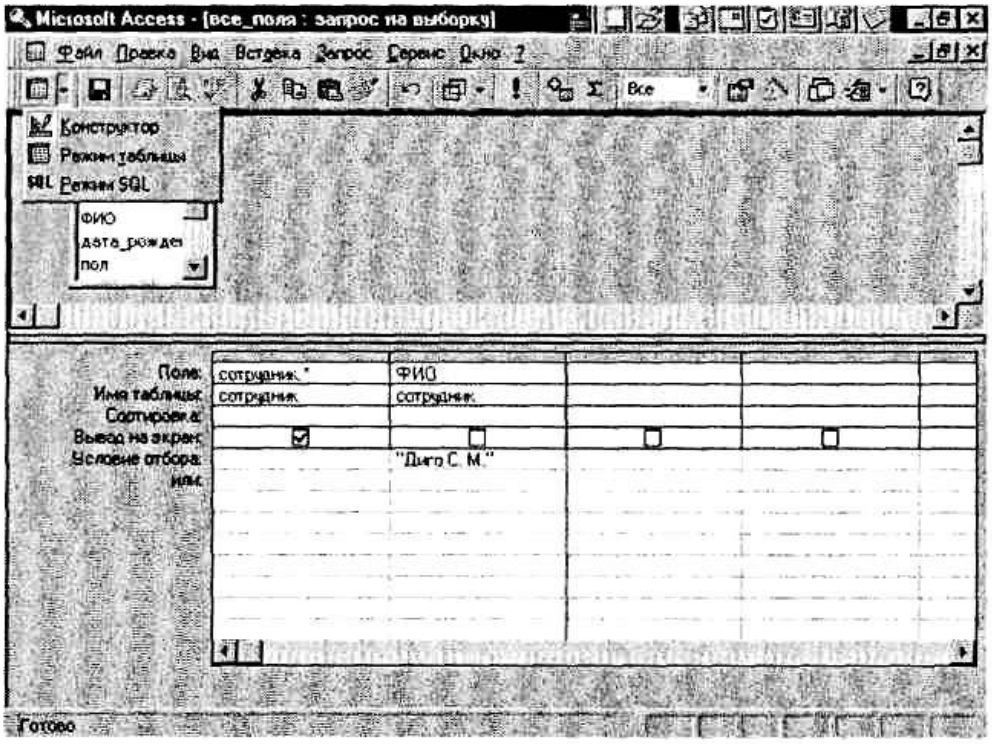


Рис. 6.15. Переключение режимов

6.2.10. Определение числа записей, выводимых в ответ

По умолчанию в ответ выводятся все отобранные записи. В Access есть возможность (выходящая за рамки двухмерных табличных языков) управлять числом записей, выводимых в ответ. Указанной возможностью можно пользоваться не только для ограничения числа записей, если отобранное множество слишком велико и для пользователя является приемлемым ограничиться определенным его подмножеством, но и для создания запросов специального вида.

В ответ на запрос можно выводить не все записи, а какое-то определенное их число. Причем это число может быть задано как в абсолютных величинах (например, выдать 10 записей, отвечающих условию отбора), так и в процентах (например, 10%). Задание числа выводимых записей часто бывает удобно сочетать в запросе с упорядочением записей. Например, если на институт выделено пять именных стипендий, можно определить средний балл студентов, упорядочить записи в порядке убывания этого поля и запросить вывод пяти записей. Если появилась возможность оказать материальную помощь определенному числу низкооплачиваемых сотрудников, то записи нужно упорядочить по возрастанию поля «Оклад».

Задать число записей, выводимых в ответ, можно по-разному. Во-первых, можно позиционироваться на свободное место в верхней части окна запросов, нажать правую клавишу мыши, в появившемся меню выбрать позицию «Свойства» и в появившемся окне Свойства запроса в строке «Набор значений» указать требуемое значение (рис. 6.16). Число записей, выводимых в ответ, можно указать не только в абсолютных величинах, но и в процентах.

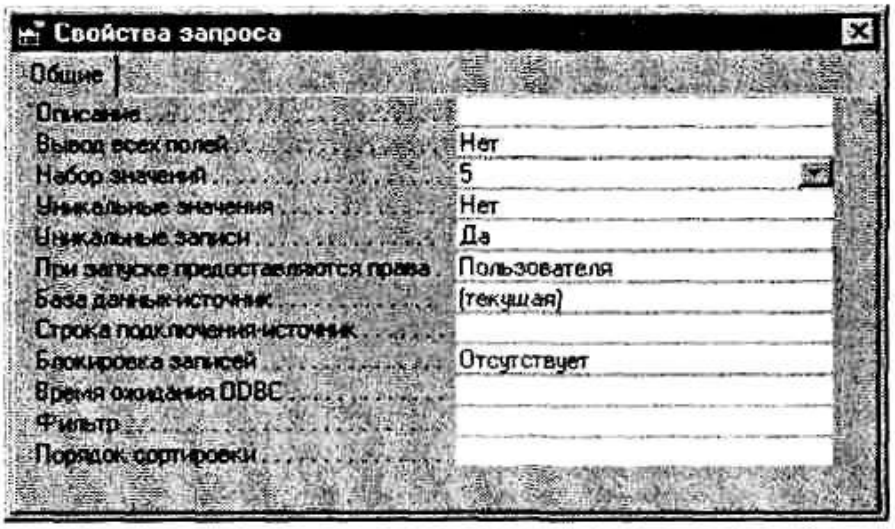


Рис. 6.16. Задание числа записей, выводимых в ответ, путем указания параметра «Набор значений» в окне **Свойства запроса**

Другим способом задания числа записей, выводимых в ответ, является использование соответствующей кнопки (рис. 6.17) на панели конструктора запросов.



Рис. 6.17. Вид кнопки **Набор значений**

Как видим, для управления выводом определенного количества записей используются «нетабличные» возможности языка запросов. В рамках табличного языка это отобразить нельзя.

6.2.11. Формирование запросов к связанным таблицам

Если была предварительно определена схема данных (см. разд. 5.2.4), то при добавлении таблиц в запрос они будут должным образом связаны. Даже если связи между таблицами не были созданы пользователем, то при добавлении в запрос двух таблиц, при условии, что они имеют поля с одинаковым или совместимым типом данных и одно из полей связи является ключевым, связи могут быть созданы автоматически. Автоматическое объединение можно разрешить или запретить. Для этого необходимо выполнить следующую последовательность шагов.

1. В меню **Сервис** выбрать команду **Параметры**.

2. Перейти к вкладке **Таблицы/Запросы**.
3. Установить/снять флажок **Автоматическое объединение**.

Параметр «Автоматическое объединение» относится только к новым запросам.

Если связи не были определены предварительно и связи не созданы автоматически, то следует задать соединение таблиц вручную (так же, как это делалось при задании схемы).

> **Внимание!** Если связь не задана (и не отменено «Автоматическое объединение»), то будет осуществляться связь каждой записи одной таблицы с каждой записью второй таблицы.

Надо осторожно относиться к формированию запросов к связанным таблицам. Как вы думаете, что будет получено в ответ на запрос, изображенный на рис. 6.18? На самом деле ответить на этот вопрос, не имея дополнительной информации, нельзя. Необходимо знать, каковы параметры объединения (если вы внимательны, то по виду линии сможете определить вид связи) и какие значения имеют свойства «Уникальные записи» и «Уникальные значения» (этого на схеме не видно). Если задано обычное («внутреннее») соединение таблиц и для свойства «Уникальные значения» задано значение «Да», то в ответ на запрос, содержащий в бланке запроса поле «ФИО» и больше ничего, будет получен список сотрудников, имеющих детей.

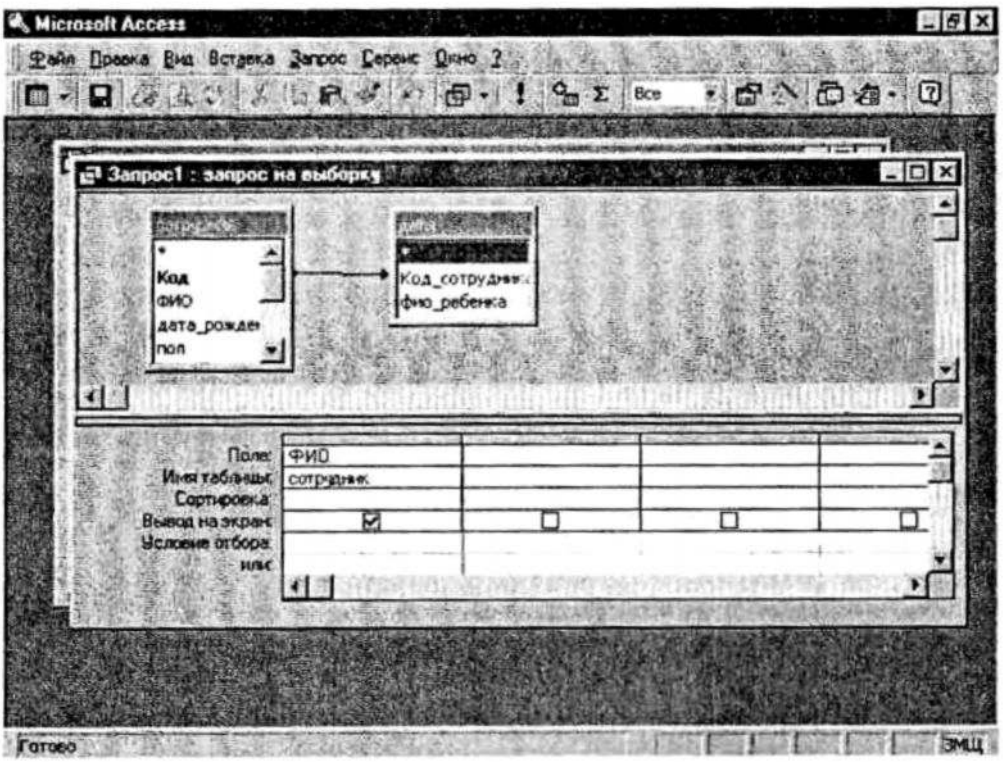


Рис. 6.18. Выполнение запросов на связанных таблицах (вариант 1)

Исходя из вышесказанного, можно дать следующие рекомендации:

1. При задании запроса удаляйте из него все таблицы, поля которых не участвуют в формировании запроса.
2. При проектировании структуры базы данных тщательно продумывайте имена, которые даете полям разных таблиц.
3. Проверяйте связи, которые система задает автоматически.

Существуют понятия внутреннего, левого и правого соединения.

В Access соединение таблиц и его тип, как правило, определяются при задании схемы. При формулировании запроса следует уточнить, какой тип объединения* был задан, и, если нужно, изменить тип соединения на тот, который необходим именно для этого запроса, поскольку тип объединения будет влиять на правильность ответа.

Изменить тип объединения в запросе можно, выделив нужную связь и нажав на правую кнопку мыши. В появившемся контекстном меню выбрать **Параметры объединения** (рис. 6.19) либо позицию меню **Вид/Параметры объединения** - появится окно **Параметры объединения** (рис. 6.20), в котором можно выбрать нужный для данного запроса тип объединения. Так, например, если требуется выдать список всех сотрудников, а для тех, кто имеет детей, - информацию о детях, то для соединения таблиц «Сотрудники» и «Дети» следует выбрать вторую альтернативу в окне **Параметры объединения**.

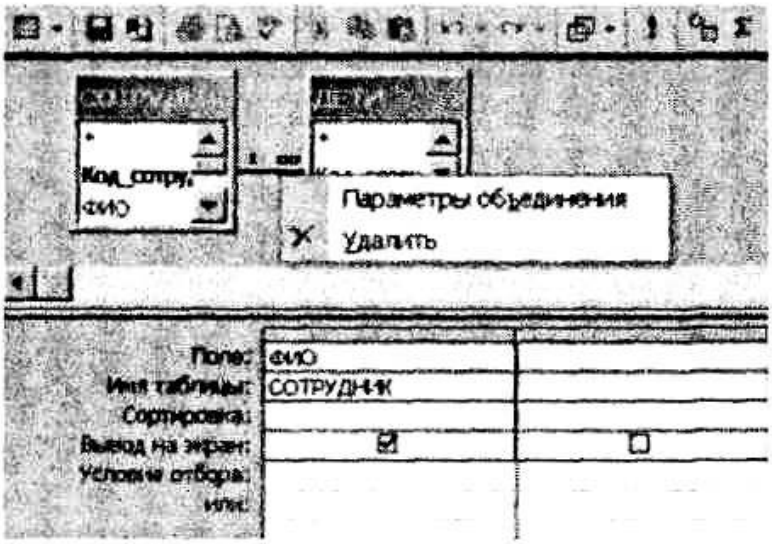


Рис. 6.19. Изменение параметров объединения при формировании запроса

Запрос, приведенный на рис. 6.21, при всей своей схожести с запросом на рис. 6.18, даст иной ответ, поскольку изменен тип объединения.

* В реляционной теории различают операции соединения и объединения. То, о чем идет сейчас речь, является реляционной операцией соединения. Но в системе Access обе эти операции называются «объединением», и когда рассматриваем, как это сделать в Access, то приходится переходить на терминологию этой системы

Возможно создание запросов, в которых таблица соединяется сама с собой (так называемое *самообъединение*). Такая ситуация может возникнуть, когда в ER-модели имеются отношения на одном и том же классе объектов. Например, для класса объектов СОТРУДНИК имеется связь «Быть руководителем». В рассматриваемом примере для отражения этой связи в таблицу «Сотрудники» введено поле «Руководитель», которое содержит код сотрудника, являющегося руководителем данного сотрудника.

Чтобы объединить две копии одной и той же таблицы в запросе, необходимо в режиме **Конструктор** запроса дважды добавить эту таблицу в запрос. Далее надо осуществить соединение таблицы с ее копией обычным путем (переместив поле из списка полей первой таблицы в соответствующее поле в списке полей другой таблицы). На рис. 6.22 изображен запрос «Для каждого из руководителей выдать список его подчиненных». Для того чтобы в результатной таблице было понятно, что означает поле «ФИО» в каждом столбце, можно переименовать эти столбцы, назвав первый «Руководитель», второй - «Подчиненный». Для этого можно щелкнуть правой клавишей мыши по соответствующему полю, в высветившемся меню выбрать позицию **Свойства**, а затем в появившемся окне **Свойства поля** в строке «Подпись» ввести требуемый заголовок столбца (рис. 6.23). Вид результатной таблицы после проведенных действий представлен на (рис. 6.24).

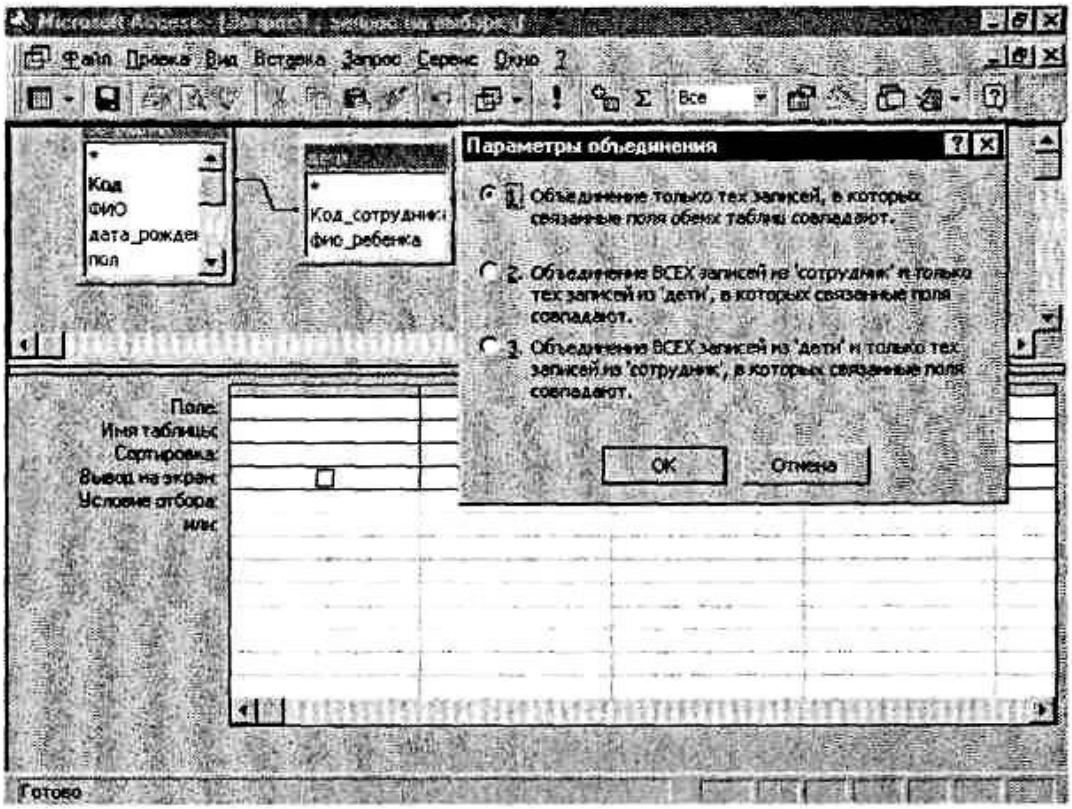


Рис. 6.20. Выбор типа объединения таблиц

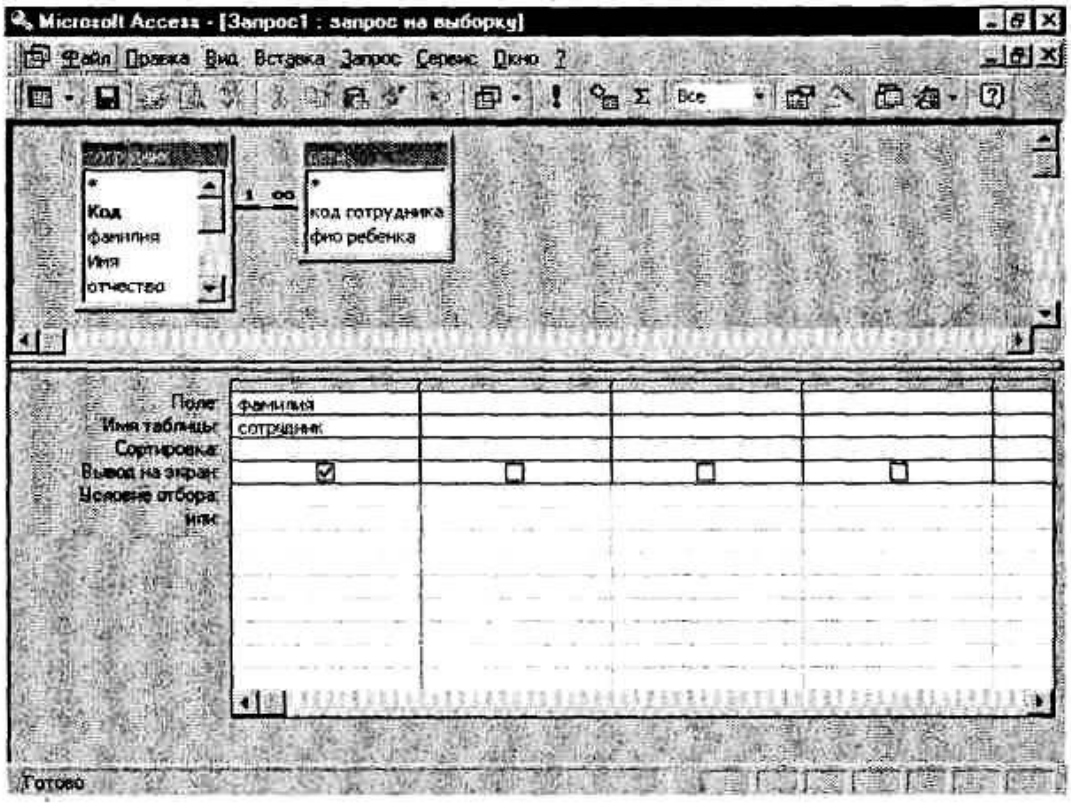


Рис. 6.21. Выполнение запросов на связанных таблицах (вариант 2)

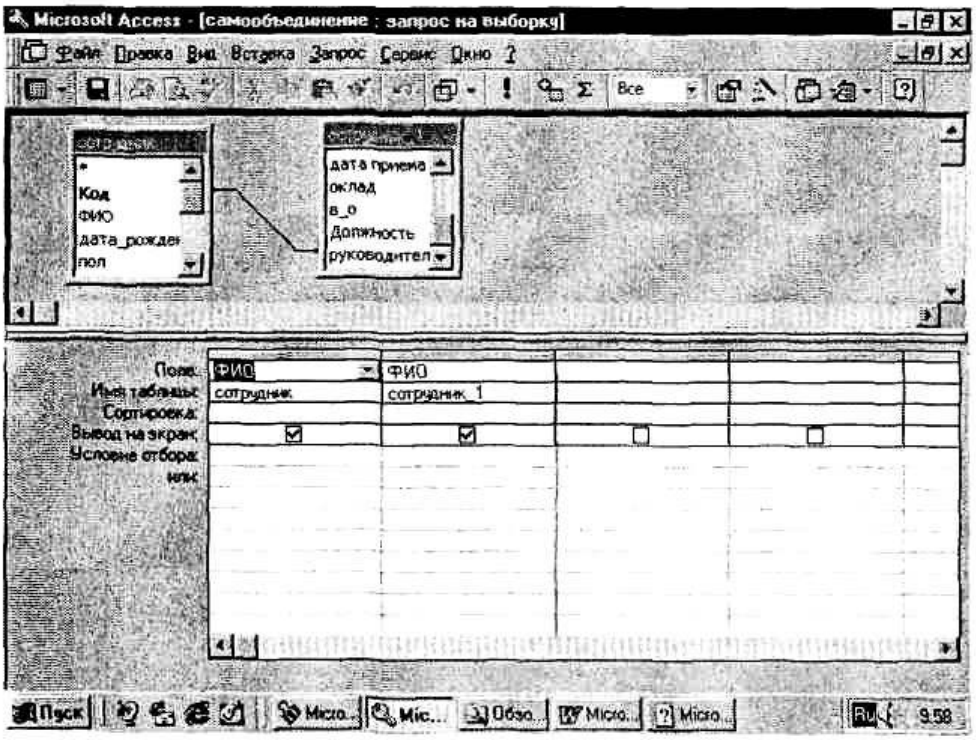


Рис. 6.22. Пример использования самообъединения таблиц в запросе

При реализации запросов на связанных таблицах по ER-модели следует уточнить характер связи между соответствующими объектами (это может повлиять на формулировку запроса) и определить, какие должны быть заданы параметры объединения для данного запроса (в случае необходимости изменить эти параметры).

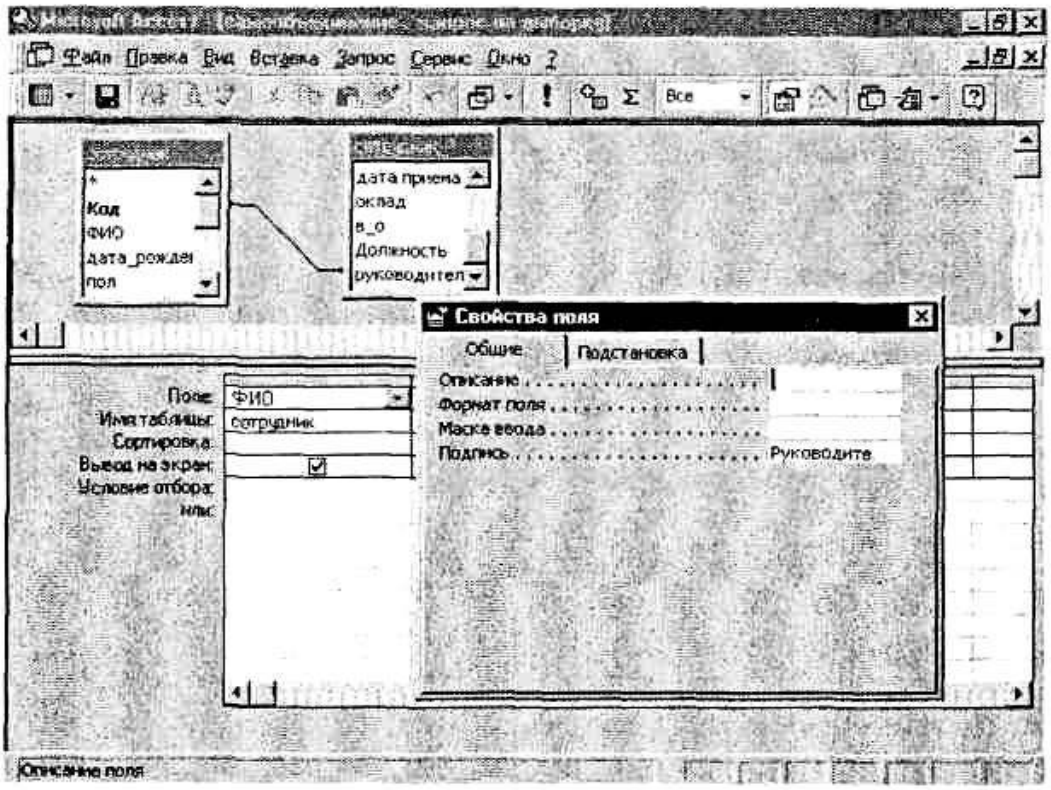


Рис. 6.23. Изменение подписи поля

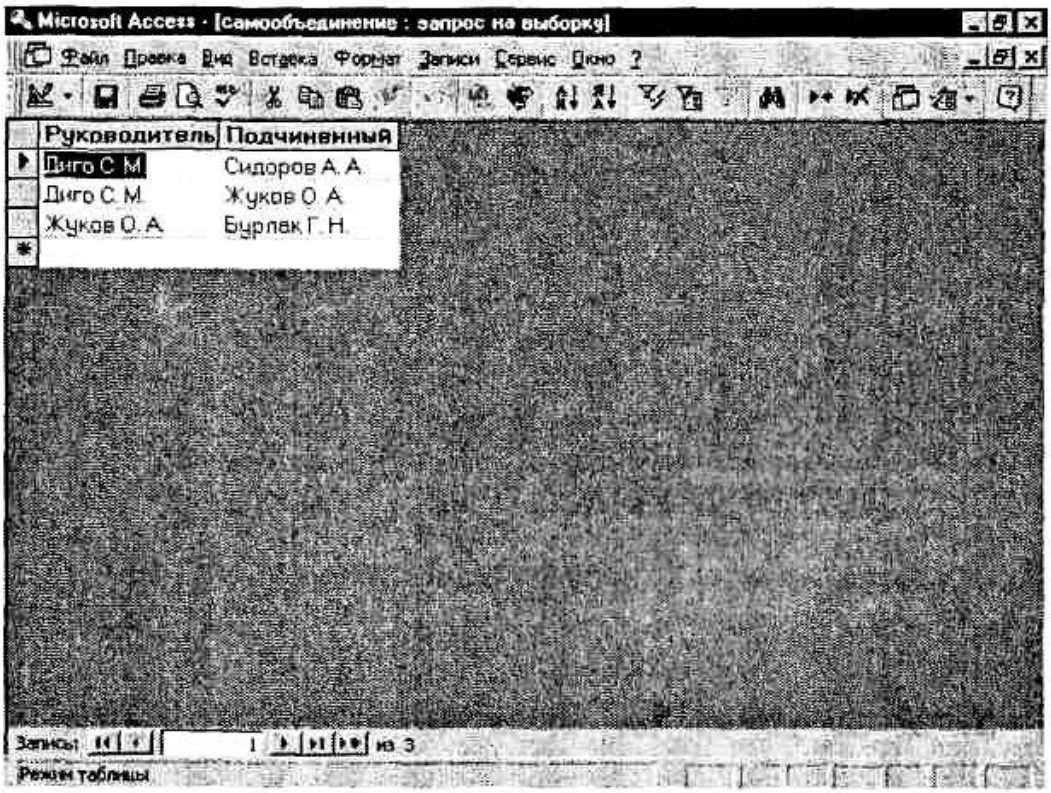


Рис. 6.24. Вид результатной таблицы
(результат запроса с самообъединением таблицы)

6.2.12. Выполнение агрегирующих операторов

Термина «обобщенный» или «агрегирующий» оператор в Access нет. Есть просто понятие «встроенные функции Microsoft Access», а среди них - «статистические функции» и «статистические функции по подмножеству».

Статистические функции - это Sum (сумма), Count (количество записей, возвращаемых запросом), Avg (среднее), Var (дисперсия) и другие, используемые для расчета итоговых значений. Статистическая функция, с помощью которой в запросе обрабатываются значения поля, может быть выбрана в ячейке строки «Групповая операция» в бланке запроса. Первоначально эта строка в бланке запроса отсутствует. Чтобы она появилась, необходимо выбрать позицию «Групповые операции» меню **Вид** или щелкнуть по кнопке со знаком «∑» на панели инструментов.

Первым полем, выводимым в ответ, должно быть поле, по которому проводится группировка, а затем - поля, над которыми проводятся вычисления. Группировка может осуществляться не только по одному полю, но и по совокупности полей. Все групповые операции, кроме Count, могут выполняться только над числовыми полями.

В строке «Групповая операция» щелчком мыши можно открыть список доступных функций и в нем сделать выбор нужной статистической функции для выполняемых над полем вычислений. На рис. 6.25 приведен пример использования групповых операций в запросе (осуществляется подсчет числа сотрудников, работающих на каждой кафедре).

В Access предварительно упорядочивать таблицу по полю (полям), по которому ведется группировка, не обязательно.

В бланке запроса задают также условия отбора, с помощью которых определяются группы, для которых вычисляются итоговые значения, записи, включаемые в вычисления, или результаты, отображаемые после выполнения расчетов. На рис. 6.26 изображен запрос, в котором условия отбора применены к вычисляемому полю («Выдать список кафедр, на которых работает меньше трех человек»).

Если предположить (а это практически всегда так), что нет кафедр, на которых не работает ни одного человека, то результат запроса будет верен и если задано внутреннее соединение, и если задано левое соединение. Но, предположим, что задается аналогичный по существу запрос «Выдать список сотрудников, имеющих меньше двух детей» на двух связанных таблицах: «Сотрудники» и «Дети». Всегда есть вероятность, что имеются сотрудники, которые не имеют детей.

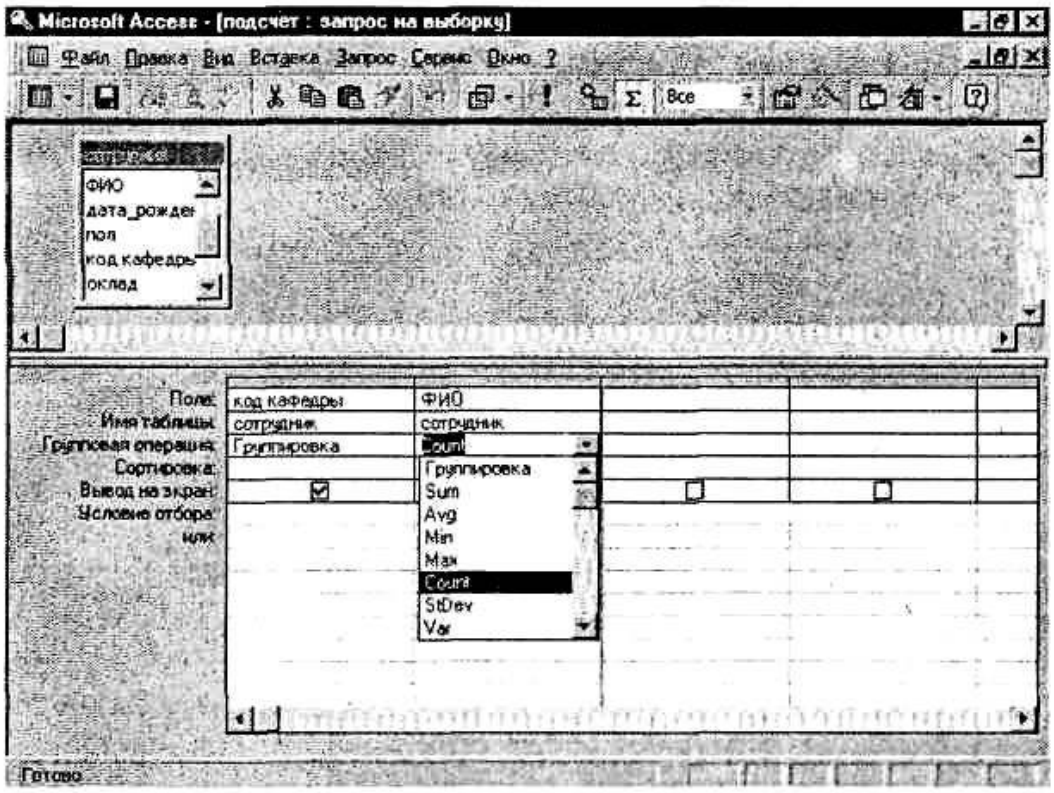


Рис. 6.25. Пример использования групповых операций в запросах

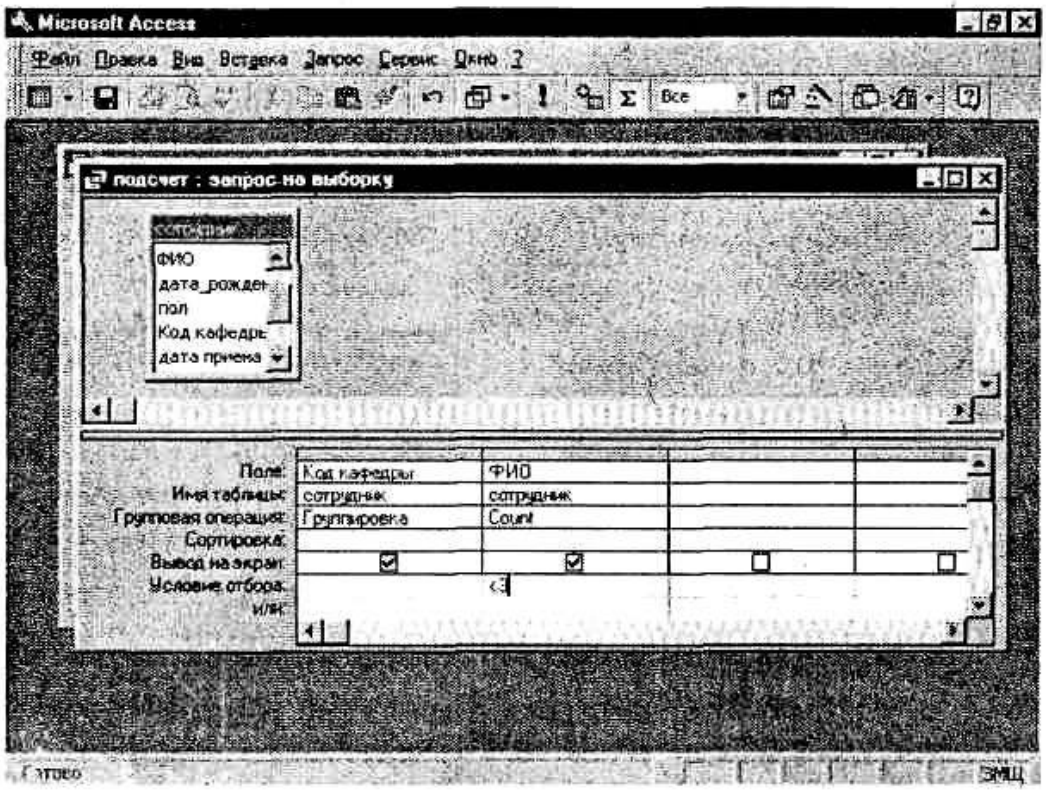


Рис. 6.26. Запрос с вычисляемым полем, используемым в условии отбора. Групповые операции

Если будет использовано внутреннее соединение (а оно задается по умолчанию), то такие сотрудники не попадут в ответ (т.е. результат ответа будет не

соответствовать действительности). На результат запроса «Выдать список сотрудников, имеющих больше двух детей» параметры объединения таблиц не окажут влияния.

Следует быть внимательными при формулировании запросов, поскольку запросы, на первый взгляд кажущиеся однотипными, могут потребовать разной реализации.

Выражения, определяющие вычисляемые поля, использующие статистические функции, могут создаваться и с помощью Мастера простых запросов, а также при создании перекрестных запросов.

6.2.13. Вычисляемые поля

При формулировании запросов можно обеспечить вывод в ответ полей, которые в исходных таблицах отсутствуют, т. е. вывод вычисляемых полей.

Например, можно перемножить значения двух полей: «Количество» и «Цена», хранящихся в БД о поставке продукции, и получить величину «Сумма», которая в БД не хранится; или вычислить дату, отстоящую на три месяца от текущей даты и т.п.

Выражения, определяемые пользователем, дают возможность выполнять действия с числами, датами и текстовыми значениями в каждой записи с использованием данных из одного или нескольких полей. Допустимые операции будут зависеть от типа полей, участвующих в выражении.

Так, для текстовых полей возможно использование оператора «+», который в этом случае воспринимается как конкатенация (соединение) строк.

Предположим, что в таблице «Сотрудник 1» имеются три отдельных поля: «Фамилия», «Имя» и «Отчество», а нужно вывести эту информацию в одном столбце. Тогда можно использовать выражение

[Фамилия] + " " + [Имя] + " " + [Отчество].

Следует обратить внимание, что, если хотя бы одно из этих трех полей будет не заполнено (причем безразлично, будет это пустое поле или там будут введены пробелы), вся строка будет пустой. В связи с этим рекомендуется вместо операции «+» использовать операцию «&». В этом случае выражение будет иметь вид

[Сотрудник1]! [Фамилия] & " " & [Сотрудник1]![Имя] & " " &
[Сотрудник 1]! [Отчество].

Для расчетов с использованием формул, определяемых пользователем, требуется создать новое вычисляемое поле прямо в бланке запроса. Вычисляемое поле создается путем ввода выражения в пустую ячейку поля в бланке запроса (рис. 6.27).

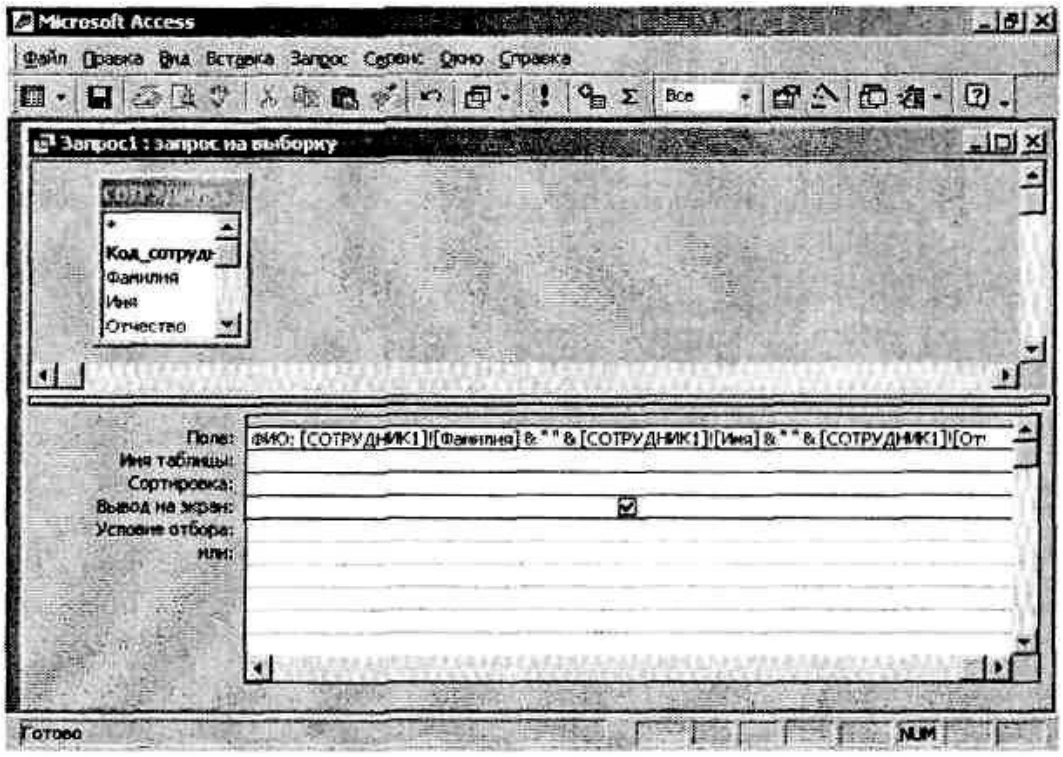


Рис. 6.27. Пример использования выражений с текстовыми полями

Для получения списка, включающего фамилии и инициалы сотрудников, можно использовать следующее выражение:

Фамилияинициалы: [Сотрудник1]! [Фамилия] & " " &
Left([Сотрудник1]![Имя];1) & "." & " " &
Left ([Сотрудник1]! [Отчество]; 1) & "."

В этом выражении использована функция Left. Второй аргумент этой функции определяет число возвращаемых символов.

Для числовых значений можно использовать любые арифметические операторы. Например, можно создать поле, в котором значение поля «Цена» умножается на значение поля «Количество», чтобы получить «Сумму к оплате».

Результаты вычислений не обязательно должны отображаться в ответе. Их можно использовать, например, в условиях отбора для определения записей, которые выводятся в ответ, но само вычисляемое поле в ответ при этом не выводится. Для того чтобы поле в ответ не выводилось, необходимо для него снять значок «V» в строке «Вывод на экран» бланка запроса. Так, на рис. 6.28 изображен запрос: «Выдать список сотрудников, работающих меньше 11 месяцев».

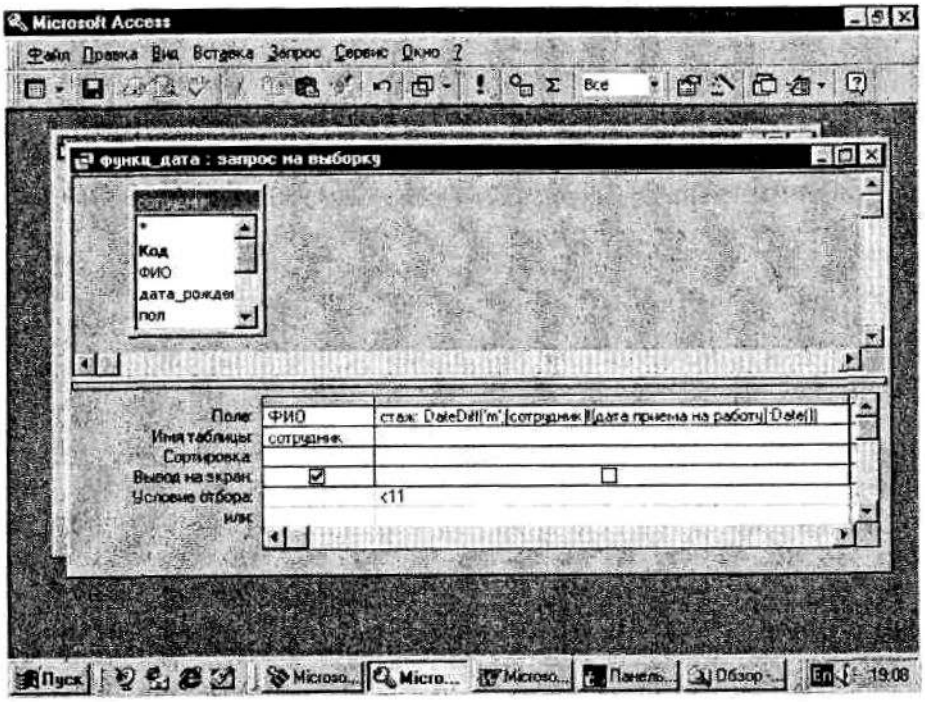


Рис. 6.28. Пример использования вычисляемого поля для отбора записей

6.2.14. Перекрестные запросы

Перекрестные запросы служат для более компактного отображения информации (схожего с изображением в электронных таблицах). В перекрестном запросе отображаются результаты статистических расчетов (такие, как суммы, количество записей, средние значения), выполненных по данным из одного поля. Эти результаты группируются по двум наборам данных в формате перекрестной таблицы. Пер-: вый набор выводится в левом столбце и образует заголовки строк, а второй - выводится в верхней строке и образует заголовки столбцов.

Например, в таблице «Сотрудники» имеются сведения о зарплате каждого сотрудника, а также признаки, на какой кафедре и в какой должности работает каждый сотрудник. Требуется для каждой кафедры определить общий фонд зарплаты, а по каждой должности - среднюю по каждой кафедре зарплату.

Для создания перекрестного запроса следует воспользоваться позицией «Перекрестный запрос» в окне **Новый запрос** {см. рис. 6.5) или выбрать соответствующую строку в меню **Запрос**. Далее необходимо выполнить ряд шагов, предлагаемых мастером по созданию перекрестных запросов.

При создании перекрестного запроса в качестве источника данных можно задать только одну таблицу (рис. 6.29). Если для реализации запроса требуются поля из разных таблиц, то надо предварительно создать вспомогательный запрос, который будет включать все требуемые поля. Так, если в создаваемом нами запросе требуется выводить название кафедры, то следует

создать запрос, базирующийся на таблицах «Кафедра» и «Сотрудник», и этот запрос выбрать в качестве источника для создаваемого перекрестного запроса. Будем выводить в ответ «Код_кафедры» и поэтому выберем таблицу «Сотрудник».

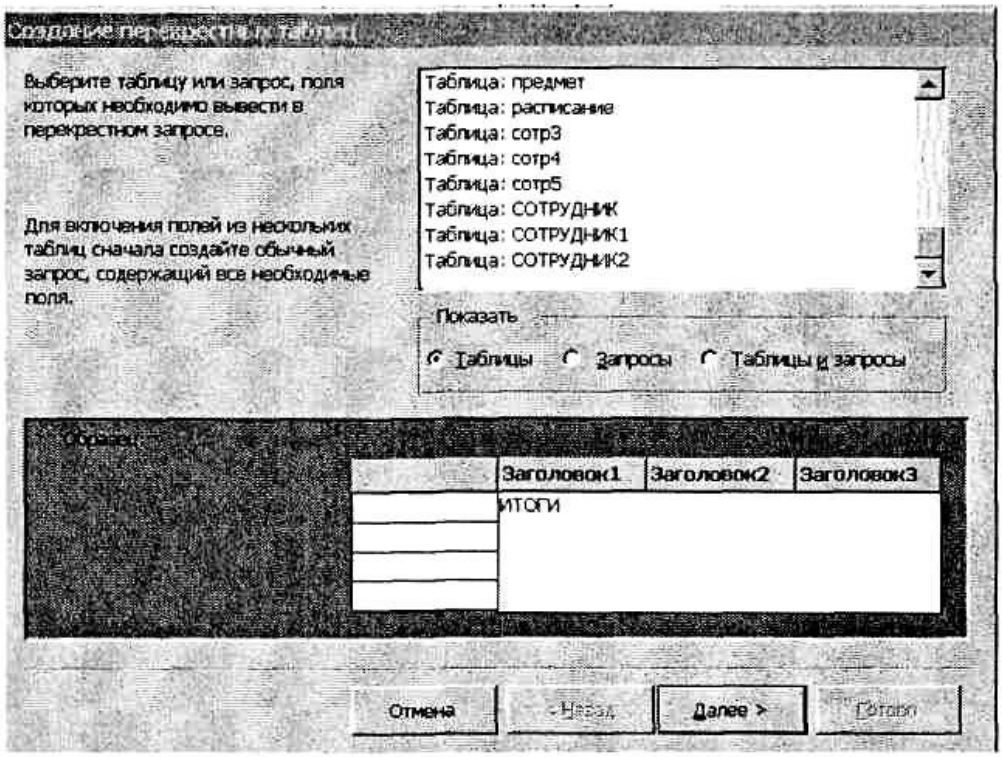


Рис. 6.29. Создание перекрестных запросов (шаг 1)

В качестве заголовков строк будет использоваться «Код_кафед-ры». Для того чтобы задать это, нужно позиционироваться на этом поле и щелкнуть по кнопке, после чего экран примет вид, представленный на рис. 6.30.

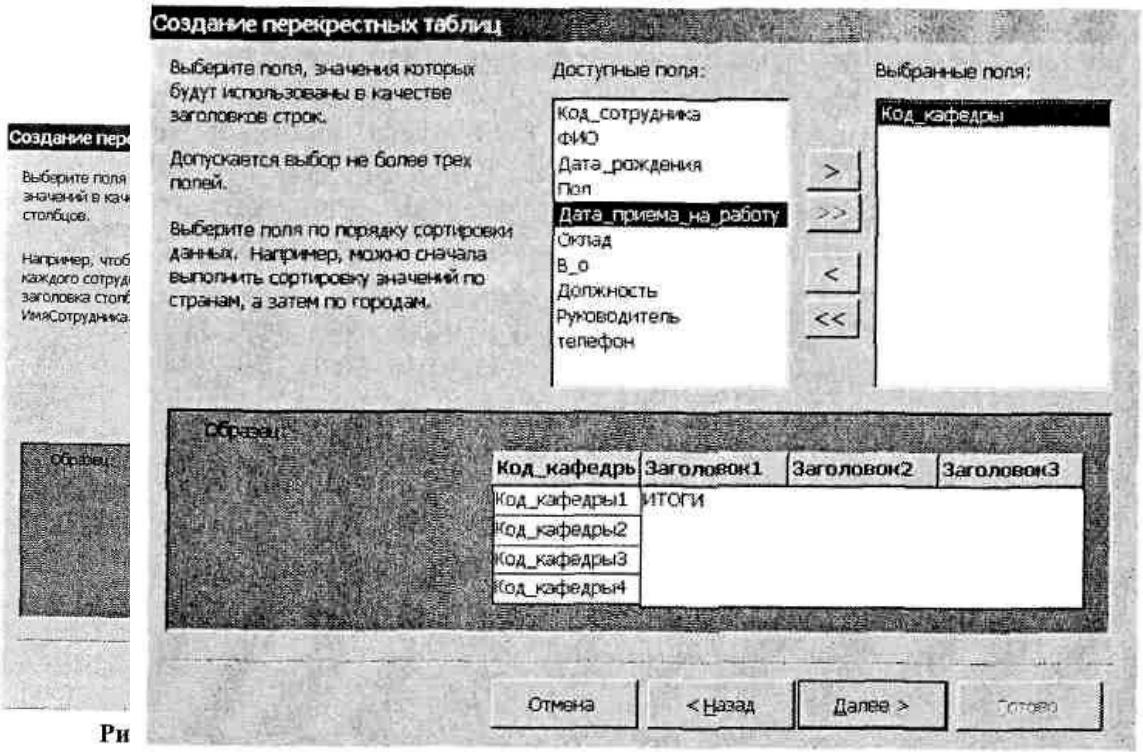


Рис. 6.30. Создание перекрестных запросов (шаг 2)

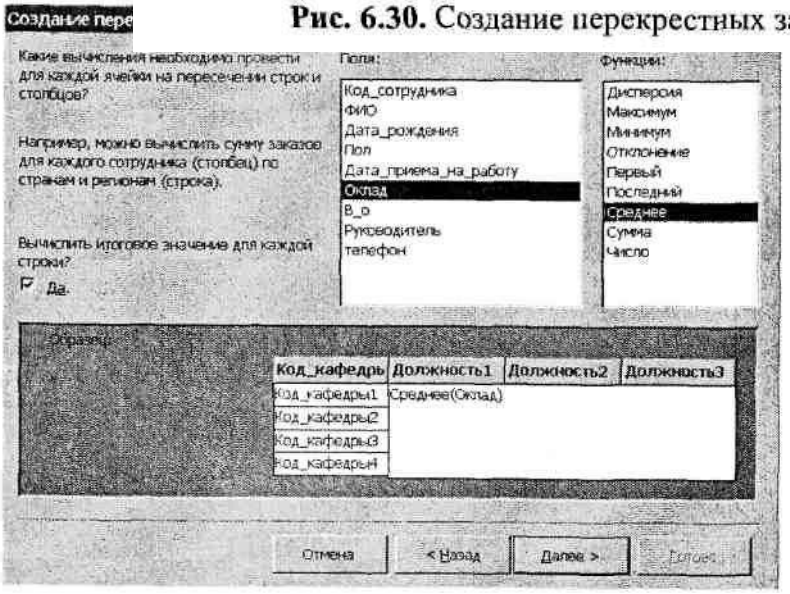


Рис. 6.32. Создание перекрестных запросов (шаг 4)

ить поле «Должность» (рис.

ке» строки и столбца и вы-
полю (рис. 6.32). На этом
ю. В том запросе, который
бщий фонд зарплаты (т.е.
ности вычислить среднюю
кцию Avg). Такой запрос с
а этом шаге функцию Avg,
лученный с помощью Мас-
запрос, который определил
дой должности в пределах

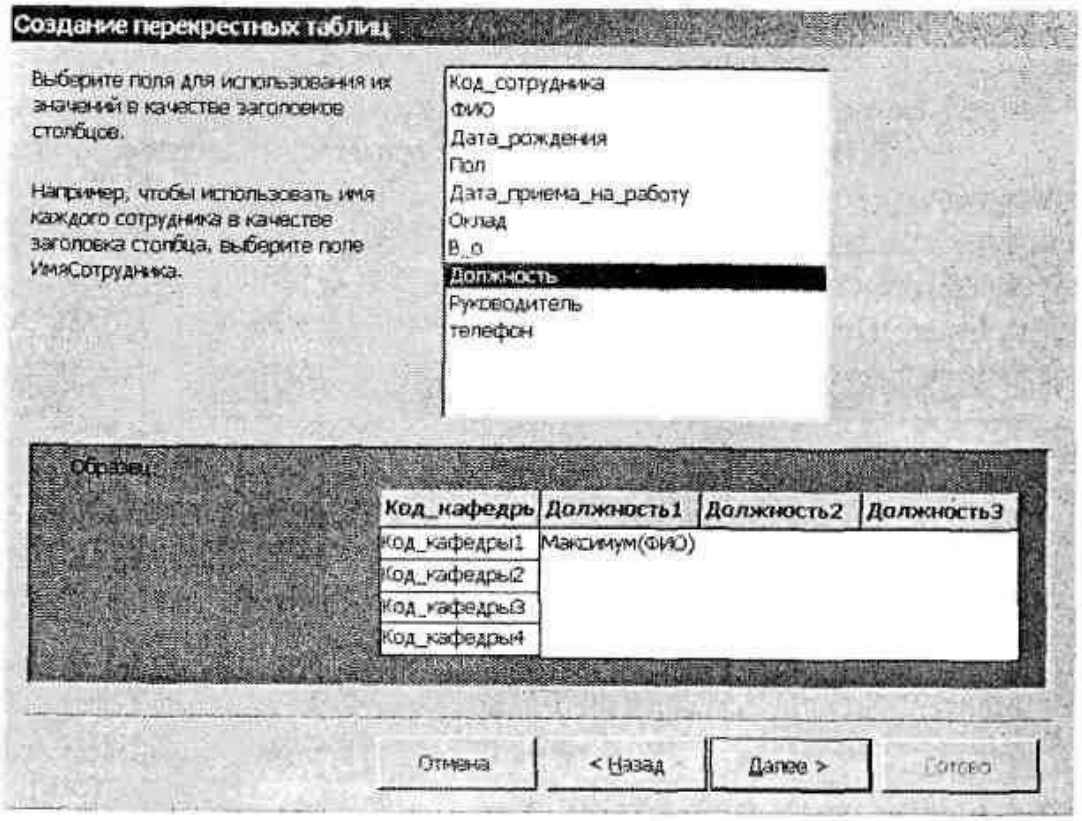


Рис. 6.31. Создание перекрестных запросов (шаг 3)

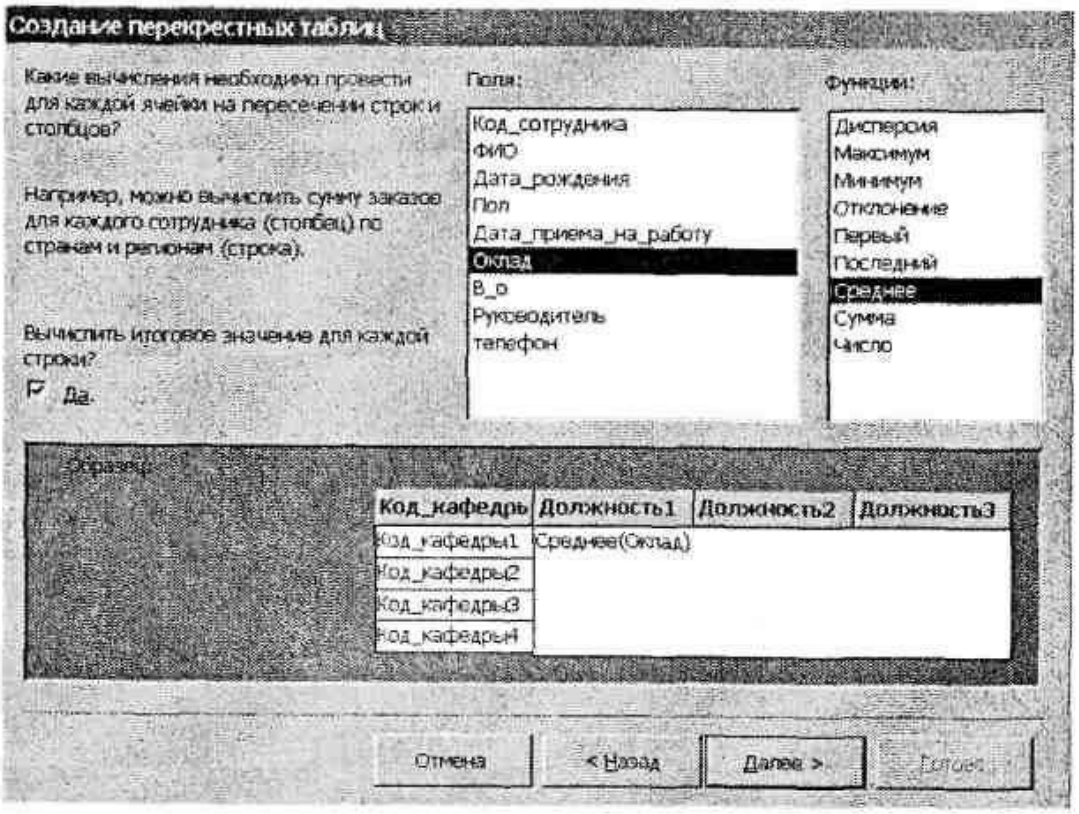


Рис. 6.32. Создание перекрестных запросов (шаг 4)

На последнем шаге создания перекрестного запроса необходимо задать имя этого запроса.

Запрос, полученный в результате использования Мастера в режиме **Конструктор** (рис.6.33), не совсем соответствует тому, что мы хотели иметь в ответе, так как в результате для каждой кафедры будет посчитана средняя зарплата, а не общий фонд, как требовалось. Для того чтобы определить требуемую сумму, необходимо сделать соответствующее изменение в «Конструкторе запросов» (рис. 6.34).



Рис. 6.33. Перекрестный запрос (вид в режиме **Конструктор**)

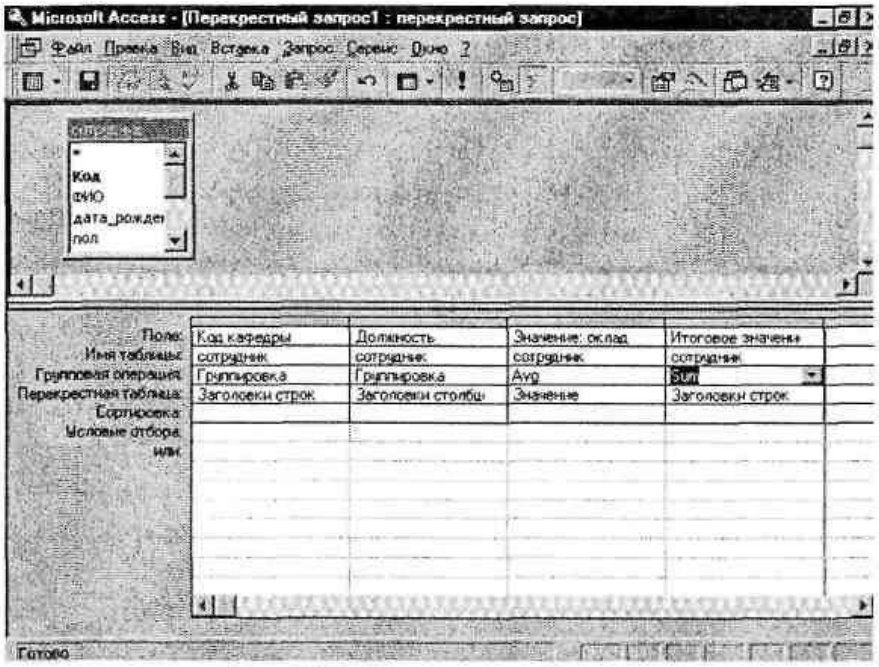


Рис. 6.34. Перекрестный запрос
(вид в режиме **Конструктор** – видоизмененный)

Код_кафедры	Итоговое значение Оклад	ассистент	доцент	профессор	старший преподаватель
Проектирования экономи	2 907,58р.	1 735,16р.		4 080,00р.	
Иностранных языков	4 129,00р.		1 610,00р.		6 648,00р.
Банковского дела	7 080,36р.		12 000,00р.		2 160,72р.
Истории	2 000,00р.	2 000,00р.			

Рис. 6.35. Перекрестный запрос (вид в режиме **Просмотр**)

Результат выполнения этого запроса представлен на рис. 6.35.

6.2.15. Создание запроса с параметрами

Если приходится часто выполнять однотипный запрос на выборку или перекрестный запрос, изменяя при этом значение какого-либо атрибута в условии отбора, то можно использовать запрос с параметрами. Запрос с параметрами не требует каждый раз вносить изменения в бланк запроса; вместо этого выводится приглашение пользователю ввести условия отбора.

Запрос с параметрами особенно удобно использовать, когда конечные пользователи, для которых создается система, не очень квалифицированные и желательно, чтобы они сами никакие элементы системы не изменяли. Запросы с параметрами также удобно использовать в качестве основы для форм и отчетов. Например, на основе запроса с параметрами можно создать ежемесячные отчеты. При выводе такого отчета на экране появится приглашение ввести месяц, за который должен быть получен отчет.

Для каждого поля, которое предполагается использовать как параметр, в «Конструкторе запросов» необходимо ввести в ячейку строки «Условие отбора» текст приглашения, заключенный в квадратные скобки. Это приглашение будет выводиться при запуске запроса. Текст подсказки должен отличаться от имени поля, но может включать его.

На рис. 6.36 представлен параметрический запрос для получения списка сотрудников заданной кафедры. При запуске этого запроса на исполнение будет появляться окно с подсказкой (рис. 6.37), в которое нужно ввести требуемое значение параметра.

Для каждого поля можно задать не только одно конкретное значение, но и диапазон значений. Так, например, для поля, в котором отображаются даты, можно вывести приглашения «Введите начальную дату:» и «Введите конечную дату:» для определения диапазона отбираемых значений. Для этого в соответствующую ячейку строки «Условие отбора» необходимо ввести выражение Between [Введите начальную дату:] And [Введите конечную дату:].

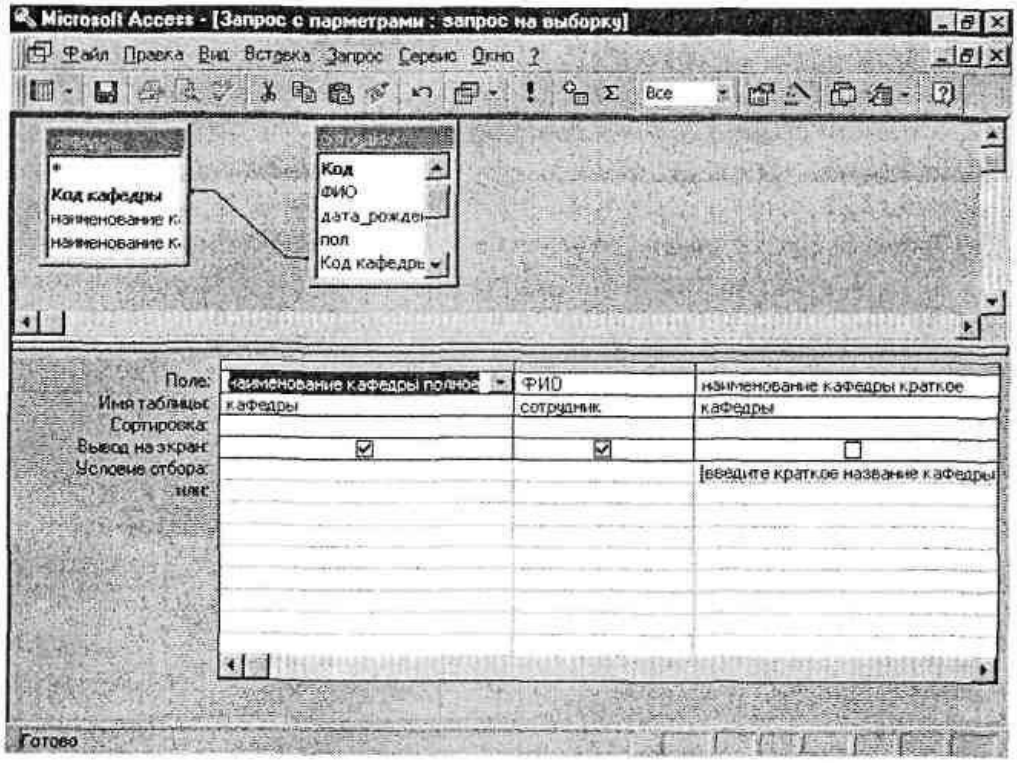


Рис. 6.36. Параметрический запрос

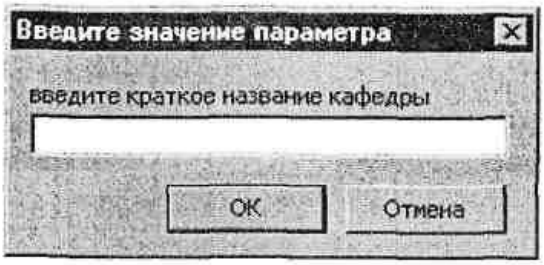


Рис. 6.37. Приглашение параметрического запроса

В качестве параметров может быть использовано не одно, а несколько полей. В этом случае для каждого поля, которое предполагается использовать как параметр, в ячейку строки «Условие отбора» вводится текст приглашения, заключенный в квадратные скобки. Эти приглашения будут последовательно выводиться при запуске запроса.

6.2.16. Корректирующие запросы

Общая характеристика. К корректирующим запросам относятся запросы на обновление (Update) и на удаление (Delete) записей, а также добавление (Append, Insert)* записей из одной таблицы или нескольких связанных таблиц в конец другой таблицы. В документации по Access к типу «запросов

* Запрос на добавление соответствует команде APPEND FROM в dBase и осуществляет вставку записей в конец таблицы; в SQL это соответствует команде Insert.

на изменение» отнесены не только перечисленные выше запросы, но еще и «запрос на создание таблицы». Последний запрос все-таки отличается от названных ранее: первый круг запросов меняет содержание существующих таблиц, а последний - создает новую таблицу, т.е. фактически меняет имеющуюся структуру базы данных. Поэтому «запрос на создание таблицы» выделен в отдельную группу.

Корректирующие запросы на обновление и удаление могут изменять как все записи таблицы, так и определенное их подмножество -это будет зависеть от условия отбора. Так, например, запрос, приведенный на рис. 6.38, увеличит зарплату всех сотрудников на 30% (так как условие отбора не задано). Запрос, изображенный на рис. 6.39, изменит зарплату одного конкретного сотрудника (*см.* условие отбора данного запроса) и установит для нее значение, указанное в запросе.

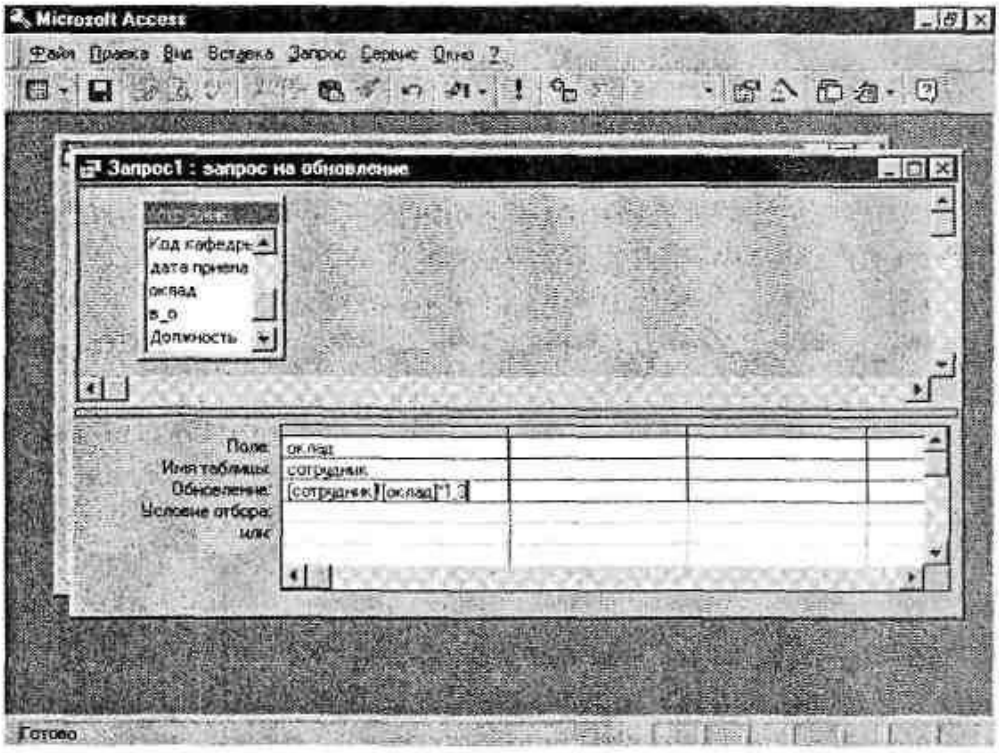


Рис. 6.38. Запрос на обновление (1)

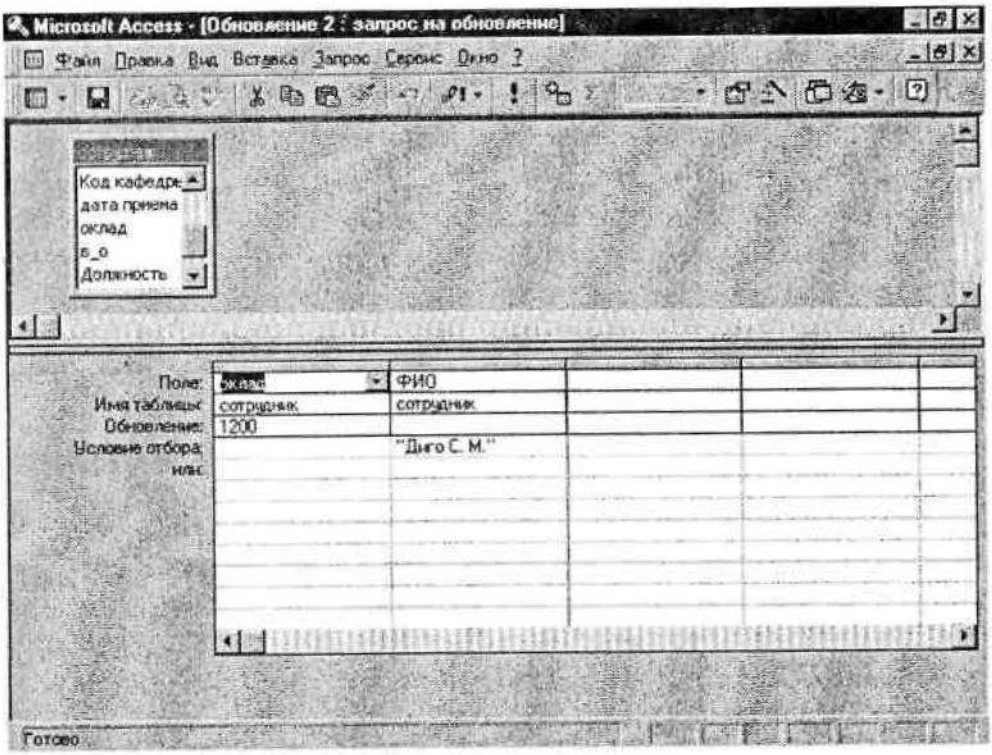


Рис. 6.39. Запрос на обновление (2)

При выполнении корректирующих запросов система осуществляет контроль ограничений целостности. Так, например, если при описании таблицы было задано ограничение на максимально допустимое значение поля «Оклад», то при выполнении запроса (см. рис. 6.38) в случае нарушения ограничения может быть выдано сообщение, подобное приведенному на рис. 6.40. Если выбрать вариант «Да», то для записей, нарушающих ограничение на значение, корректировка вообще выполнена не будет. В нашем примере такой выбор будет неправильным, поскольку в этом случае у одних сотрудников зарплата будет повышена, а у других - нет. Причем потом разобраться, у кого она была повышена, а у кого - нет, будет практически невозможно. В рассматриваемой ситуации следует отказаться от выполнения запроса, скорректировать при необходимости ограничение на значение и только после этого выполнить запрос.

Для создания корректирующего запроса надо в режиме **Конструктор запроса** выбрать соответствующую позицию в меню **Запрос** (либо нажать стрелку рядом с кнопкой **Тип запроса** на панели инструментов). В приведенных выше примерах выбиралась позиция меню **Обновление**.



Рис. 6.40. Контроль ограничений целостности при выполнении корректирующих запросов

Чтобы просмотреть обновляемые записи перед выполнением запроса, можно щелкнуть по кнопке **Вид** на панели инструментов. Выводимый список будет содержать старые значения полей отобранных в запросе записей.

> **Внимание!** Следует быть очень внимательным перед выполнением корректирующих запросов (недаром в списке запросов перед их именем стоит восклицательный знак), так как каждый их запуск на выполнение изменяет содержимое таблиц и отменить результат выполнения нельзя. В связи с этим рекомендуется сохранять резервные копии изменяемых таблиц. Это позволит восстановить ошибочно скорректированные записи.

Запрос на обновление. Для создания запроса на обновление необходимо сначала в режиме **Конструктор** выбрать таблицу(ы), поля которой будут корректироваться, а затем изменить тип запроса, выбрав позицию **Обновление** в меню **Запрос** (либо нажать стрелку рядом с кнопкой **Тип запроса** на панели инструментов и выбрать соответствующий тип). При этом вид бланка запроса изменится по сравнению с запросом на выборку: в бланке отсутствуют строки «Сортировка» и «Вывод на экран», но зато появилась строка «Обновление». В бланк запроса включаются те поля, значения которых будут изменяться. В строке «Обновление» соответствующего поля записывается его новое значение или выражение для его вычисления. При создании выражения можно воспользоваться построителем. Если обновление касается не всех записей таблицы, а только некоторых из них, то нужно обычным обра-

зом задать условия отбора. С целью убедиться, что условия отбора заданы верно, рекомендуется сначала посмотреть результаты отбора. Это можно сделать двумя способами:

1. просмотреть запрос в режиме таблицы;
2. выполнить запрос как запрос на выборку.

«Открытие» запроса на обновление означает изменение значений в базе данных. Каждое «Открытие» запроса на обновление будет означать обновление уже обновленных данных.

Запрос на удаление. Его используют для удаления группы записей, отбираемых с помощью указанных пользователем условий отбора. Сначала необходимо задать условие отбора. Перед выполнением запроса на удаление рекомендуется предварительно просмотреть результаты отбора.

Для создания запроса на удаление следует в режиме **Конструктор запроса** выбрать соответствующую позицию «Удаление» в меню **Запрос** (либо нажать стрелку рядом с кнопкой **Тип запроса** на панели инструментов).

На рис. 6.41 изображен запрос, приводящий к удалению из таблицы «Сотрудник» записей, соответствующих стажерам.



Рис. 6.41. Запрос на удаление записей

Внимание! Если в запросе на удаление не указать условие отбора, то будут удалены все записи. В некоторых случаях запрос-удаление приводит к удалению записей в таблицах, не включенных в запрос. Это случается, если в запрос включена таблица, находящаяся на стороне «один» отношения «один ко многим», а целостность данных обеспечивается параметром «Каскадное удаление». При удалении записей из этой таблицы будут также удалены связанные записи из таблицы, находящейся на стороне «многие».

В многотабличном запросе в бланк запроса QBE необходимо переместить символ звездочки (*) для каждой таблицы, из которой будут удаляться записи, и поля, используемые для задания условий отбора.

Запрос на добавление. Он добавляет группу записей из одной или нескольких таблиц (таблиц-источников) в конец другой таблицы (результатной таблицы). Для задания запроса такого типа следует сначала создать запрос, содержащий таблицу (таблицы), записи из которой необходимо добавить в другую таблицу. Затем в режиме **Конструктор запроса** нужно нажать стрелку рядом с кнопкой **Тип запроса** на панели инструментов и выбрать команду **Добавление** (либо выбрать соответствующую позицию в меню **Запросы**). На экране появится диалоговое окно **Добавление** (рис. 6.42). В поле «Имя таблицы» следует ввести имя таблицы, в которую необходимо добавить записи.

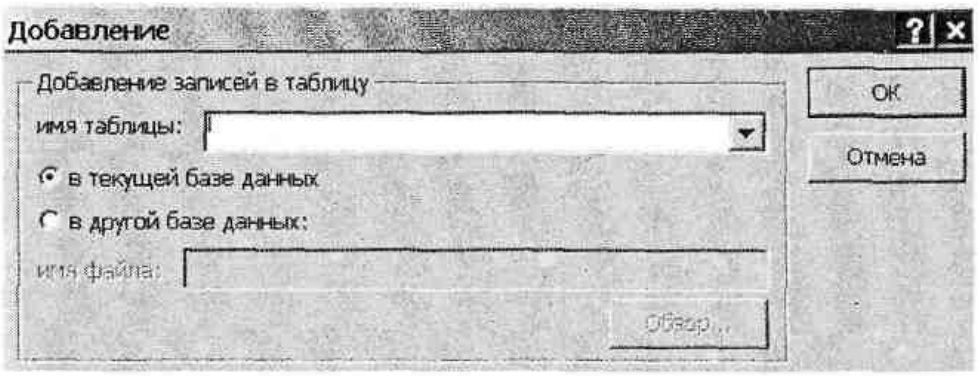


Рис. 6.42. Запрос на добавление

Таблица, в которую осуществляется добавление, может быть как в той же базе данных, так и в другой, причем это не обязательно должна быть база данных Access (это может быть Microsoft FoxPro, Paradox или dBASE, а также база данных SQL).

Из списка полей таблиц-источников в бланк запроса необходимо переместить поля, которые должны входить в состав добавляемых записей, а также те, которые будут использованы при определении условия отбора.

Если все поля таблицы-источника должны быть перенесены в целевую таблицу и поля в таблице-источнике и целевой таблице имеют одинаковые имена, то можно просто переместить с помощью мыши символ «звездочка» (*) в бланк запроса. Однако при работе с репликой базы данных добавлять придется все поля. Кроме того, при использовании символа «звездочка» (*), даже если структуры обеих таблиц полностью совпадают, могут возникнуть проблемы с ключами (если ключевое поле имеет тип «Счетчик», то для автоматического добавления значений счетчика не следует при создании запроса перемещать поле счетчика в бланк запроса).

Если в обеих таблицах выделенные поля имеют одинаковые имена, соответствующие имена автоматически вводятся в строку «Добавление». Если имена полей двух таблиц отличаются друг от друга, в строку «Добавление» необходимо ввести имя поля целевой таблицы, которое соответствует полю таблицы-источника.

Следует быть внимательным при создании запроса на добавление записей в таблицу. Все поля целевой таблицы, которые имеют либо свойство «Обязательное поле», либо заданные «условия на значение», должны быть включены в состав добавляемых полей, иначе ни одна запись добавлена не будет по причине нарушения ограничений целостности.

6.2.17. Запрос на создание таблицы

Запрос на создание таблицы фактически означает запоминание результата запроса в таблице. Чтобы использовать такую возможность, необходимо создать запрос, результат которого следует поместить в новую таблицу. Затем в режиме конструктора запроса нужно выбрать **Тип запроса/Создание таблицы** (рис. 6.43).

На экране появится диалоговое окно **Создание таблицы** (рис. 6.44). В поле «Имя таблицы» необходимо ввести имя таблицы, в которую будут переноситься данные.

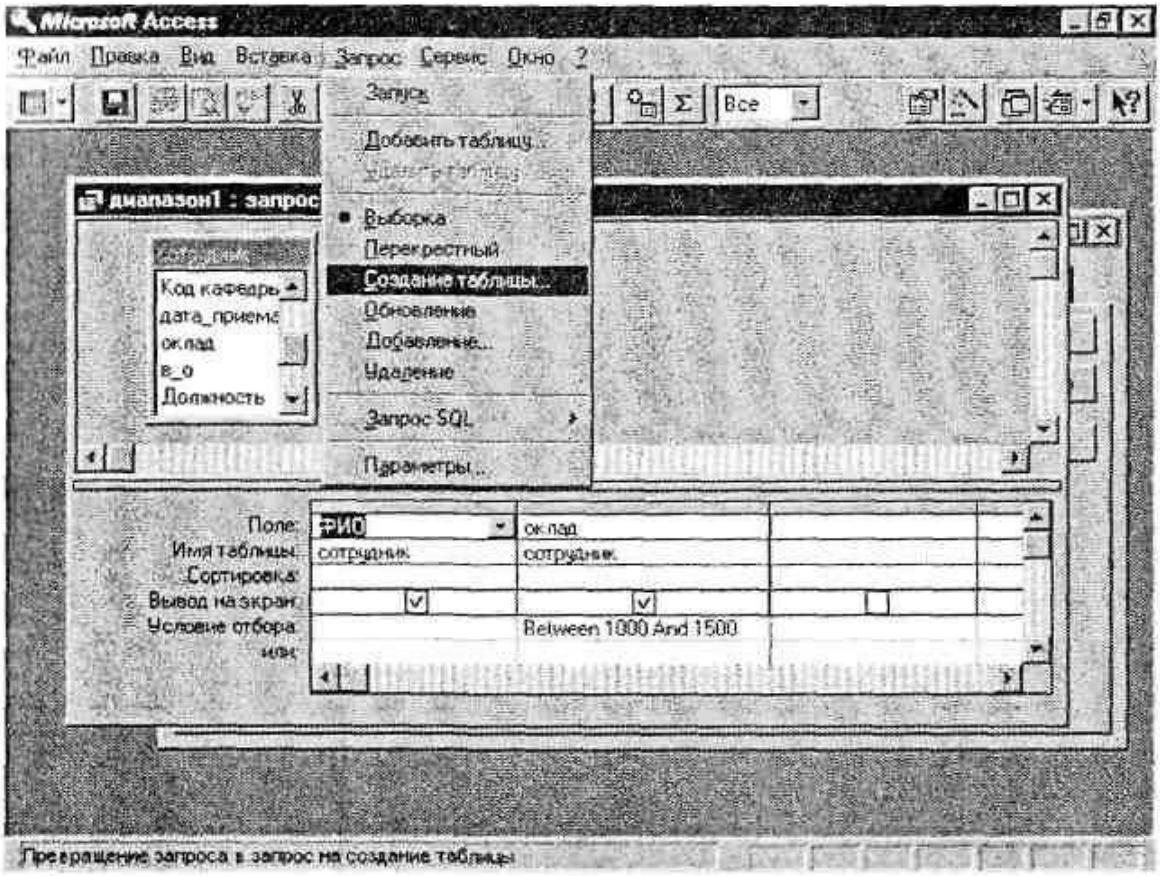


Рис. 6.43. Создание таблицы путем запоминания результата запроса (шаг 1)

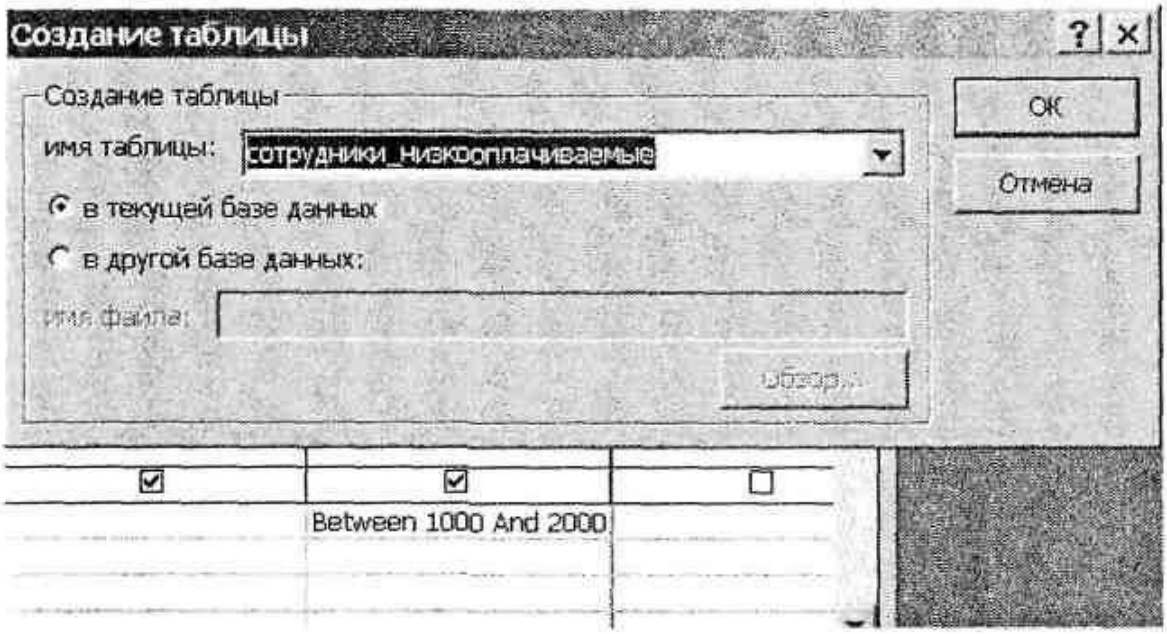


Рис. 6.44. Создание таблицы путем запоминания результата запроса (шаг 2)

6.2.18. Специальные запросы

В Access есть возможность с помощью Мастеров формировать специфические запросы: записи без подчиненных, дублирование значений полей, а также рассмотренные выше перекрестные запросы. Использование этих Мастеров позволяет достаточно просто формулировать сложные запросы.

Поиск записей, не имеющих подчиненных. Необходимость поиска записей, не имеющих подчиненных, возникает довольно часто, и не только для проверки целостности базы данных. В нашем примере воспользуемся такой возможностью для определения списка сотрудников, не имеющих детей.

Для того чтобы воспользоваться возможностью поиска записей, не имеющих подчиненных, можно выбрать Мастер «Записи без подчиненных» в окне **Новый запрос** (см. рис. 6.5).

Затем следует выбрать основную таблицу (рис. 6.45) в паре «основная - подчиненная». В нашем случае это таблица «Сотрудник». Основная и подчиненная таблицы должны быть предварительно связаны в схеме данных.

Далее выбирается подчиненная таблица. В нашем случае это таблица «Дети» (рис. 6.46).

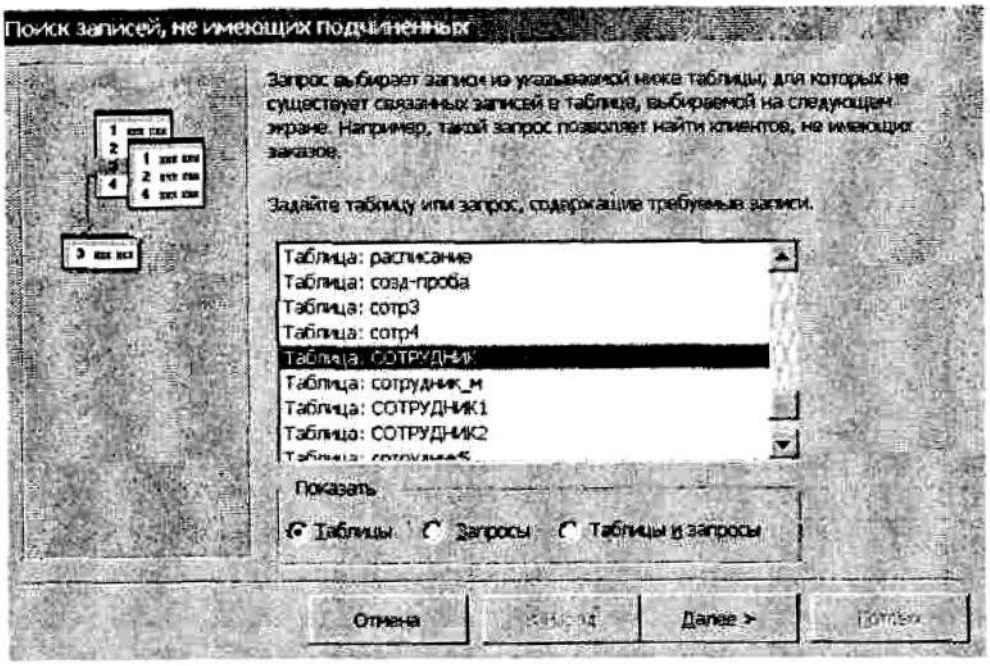


Рис. 6.45. Поиск записей, не имеющих подчиненных.
Шаг 1 – выбор основной таблицы

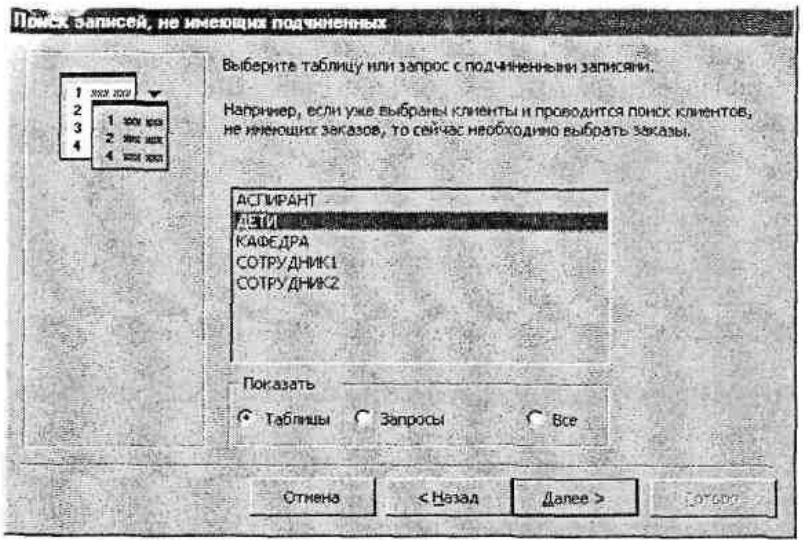


Рис. 6.46. Поиск записей, не имеющих подчиненных.
Шаг 2 – выбор подчиненной таблицы

На следующем шаге определяются поля, по которым связаны выбранные таблицы (рис. 6.47).

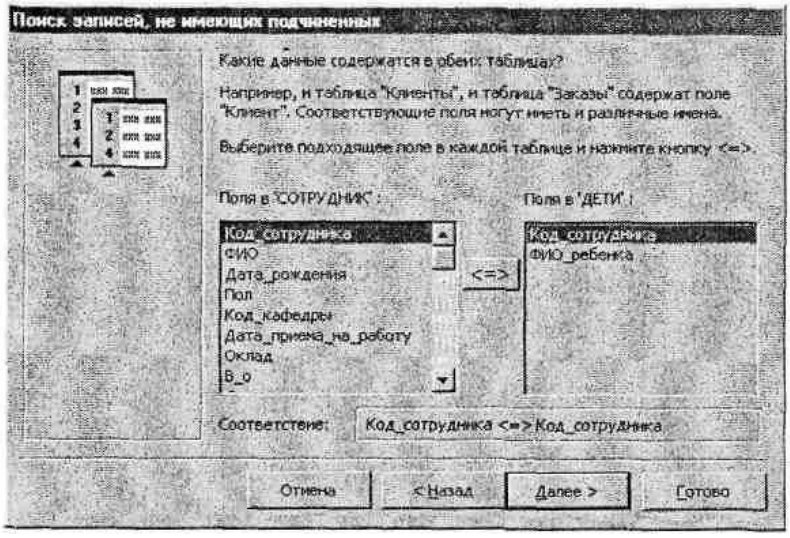


Рис. 6.47. Поиск записей, не имеющих подчиненных.
Шаг 3 – определение полей связи

Затем выбираются поля, которые должны войти в ответ (рис. 6.48). Поскольку необходим просто список сотрудников, в ответ выводится только поле «ФИО».

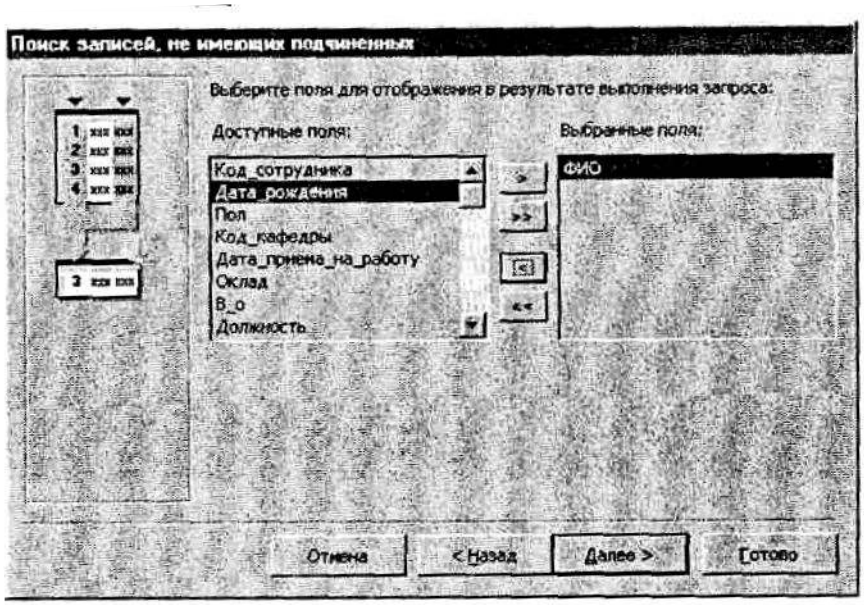


Рис. 6.48. Поиск записей, не имеющих подчиненных.
Шаг 4 – выбор полей, выводимых в ответ

В завершение следует задать имя созданного запроса (рис. 6.49).

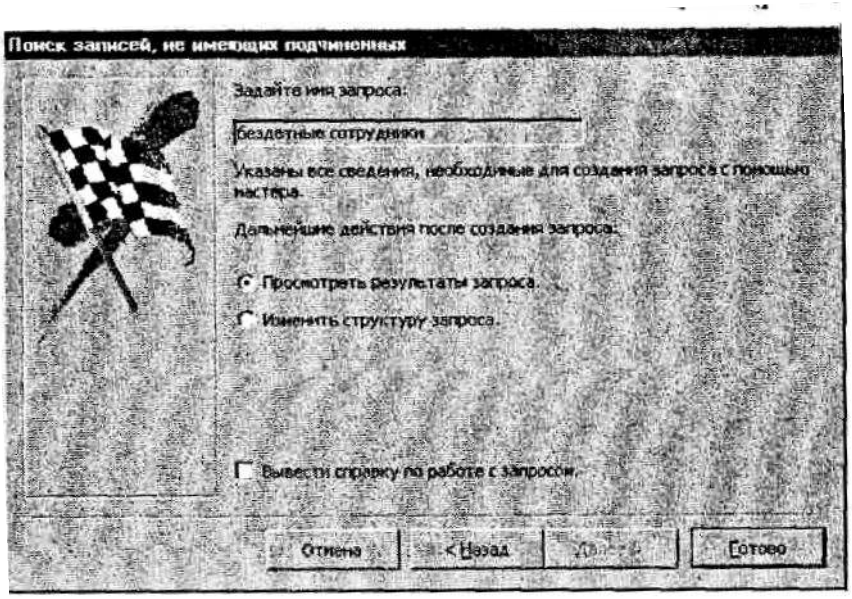


Рис. 6.49. Поиск записей, не имеющих подчиненных.
Шаг 5 – задание имени запроса

Рассматриваемый запрос можно было задать и не пользуясь Мастером. Посмотрим, как выглядит созданный нами запрос в режиме **Конструктор** (рис. 6.50). Создание подобных запросов и в этом режиме не представляет особых трудностей. Но следует обратить внимание, что при связи основной и зависимой таблиц обязательно должно быть определено так называемое «левое соединение» (т.е. для связи в «параметрах объединения» необходимо выбрать вторую возможность - «объединение всех записей из первой таблицы и

только тех записей из второй таблицы, в которых связанные поля совпадают). В противном случае список окажется пустым.

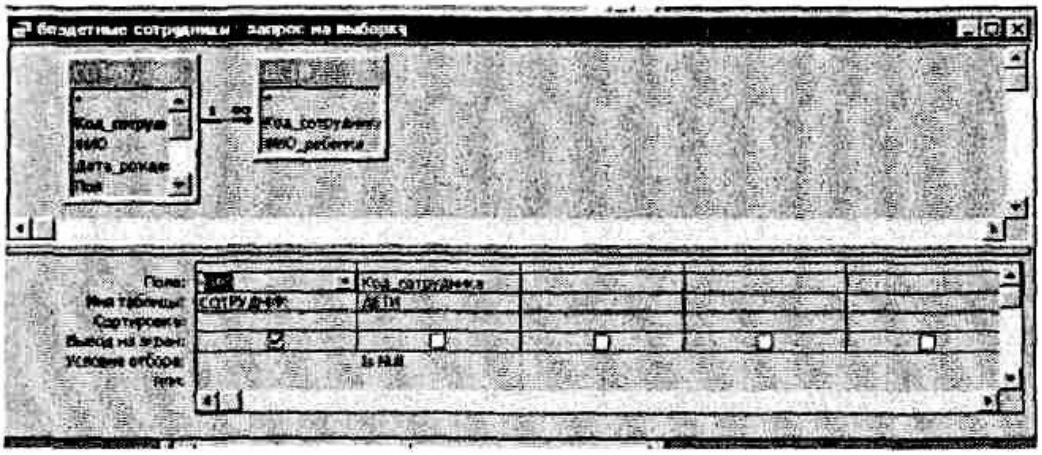


Рис. 6.50. Вид запроса в режиме **Конструктор**

Повторяющиеся записи. Так же как и в случае «поиска записей, не имеющих подчиненных», необходимость в поиске «повторяющихся записей» возникает не только для проверки целостности базы данных, но и для выполнения специфических поисковых запросов.

Название «повторяющиеся записи» не совсем точно отражает суть этого типа запросов. Имеется в виду поиск записей, имеющих одинаковое значение в заданных полях. Например, при составлении расписания требуется проверить, что для преподавателя не назначили несколько занятий в одно и то же время.

Для того чтобы осуществить поиск записей, не имеющих, подчиненных, можно выбрать Мастер «Записи без подчиненных» в окне **Новый запрос** (см. рис. 6.5).

На первом шаге создания данного запроса надо выбрать таблицу или запрос, в которых нужно найти записи с повторяющимися значениями (рис. 6.51). Для нашего примера это таблица «Расписание».

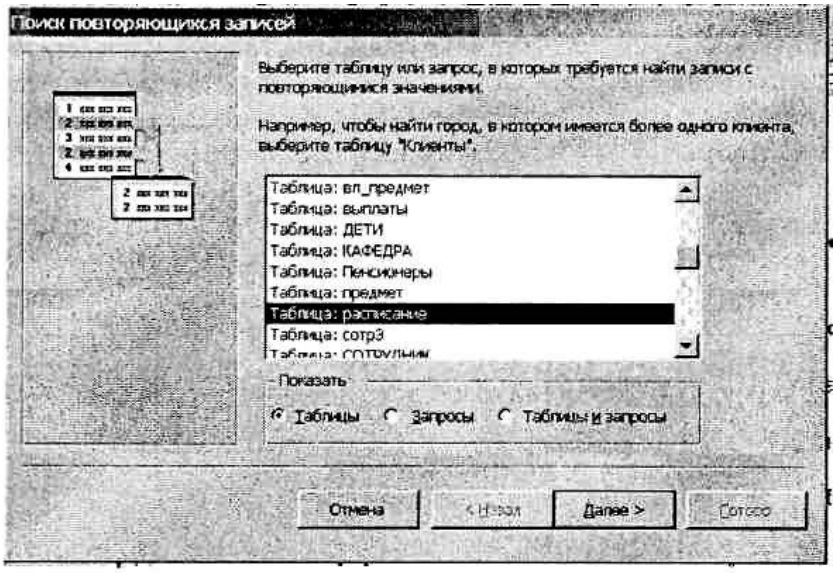


Рис. 6.51. Вид окна **Поиск повторяющихся записей** (шаг 1)

На следующем шаге необходимо определить поля, дублирование значений которых будет выявляться. Для этого следует позиционироваться на соответствующее поле и щелкнуть по кнопке. Для нашего примера это «Код_ сотрудника», «Дата», «Время» (рис. 6.52).

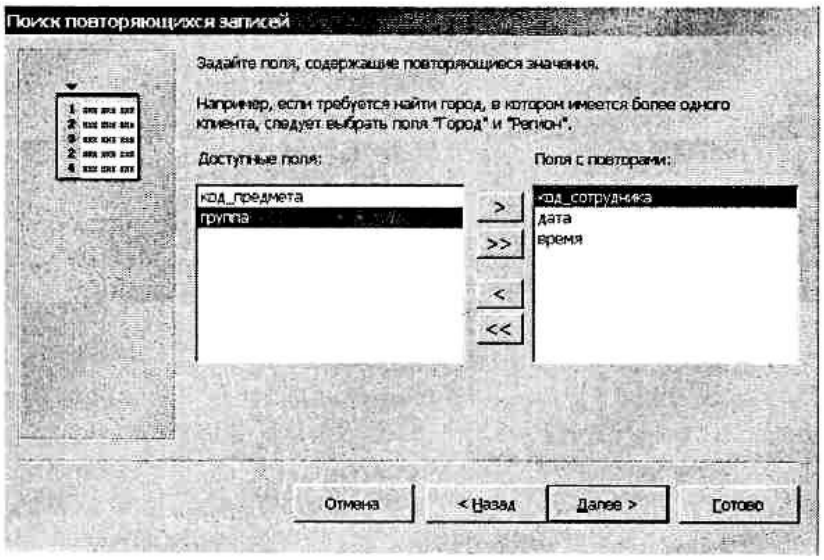


Рис. 6.52. Вид окна **Поиск повторяющихся записей** (шаг 2)

На следующем шаге (рис. 6.53) нужно выбрать те поля, которые будут также отображаться в ответе наряду с полями, выбранными на предыдущем этапе. Мы выберем оба оставшихся поля

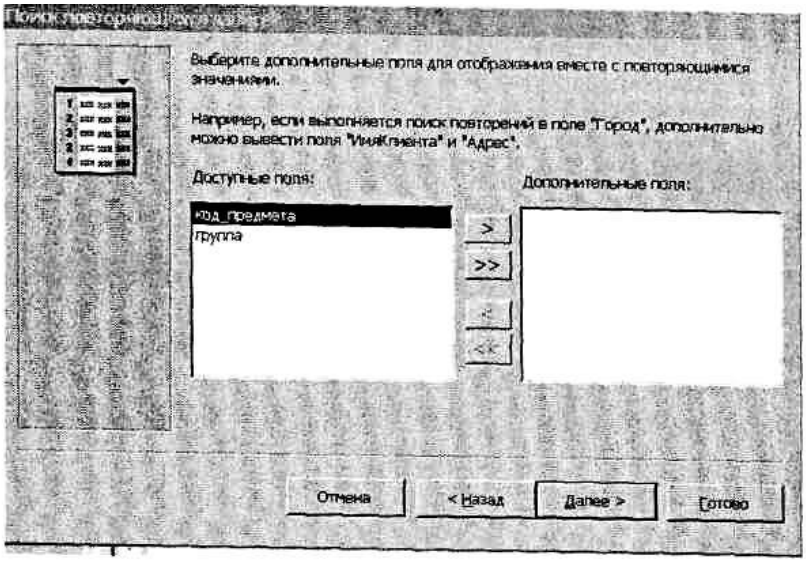


Рис. 6.53. Вид окна **Поиск повторяющихся записей** (шаг 3)

На последнем шаге (рис. 6.54) следует задать имя создаваемого запроса

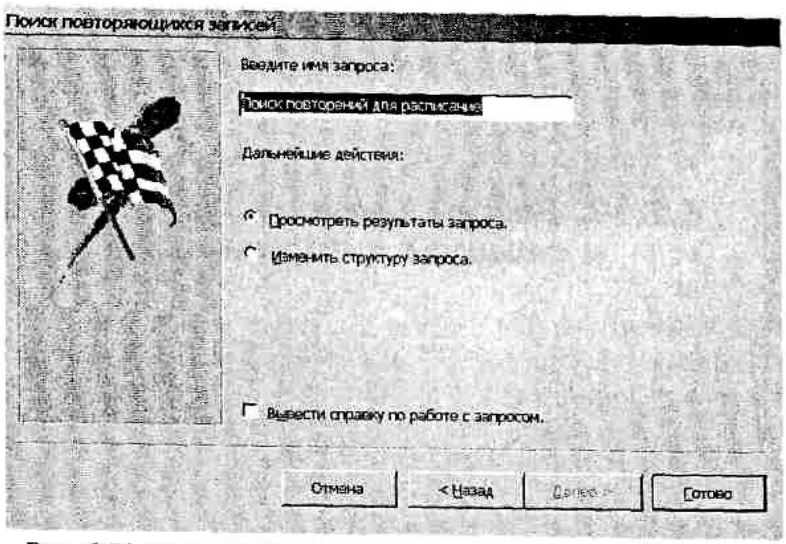


Рис. 6.54. Вид окна **Поиск повторяющихся записей** (шаг 4)

Вид полученного запроса в режиме **Конструктор** представлен на рис. 6.55. В строке «Условие отбора» в поле «Код_сотрудника» записано следующее выражение:

In (SELECT [кодсотрудника] FROM [расписание] As Tmp GROUP BY [код_сотрудника],[дата],[время] HAVING Count(*)>1 And [дата] = [расписание].[дата] And [время] = [расписание].[время])

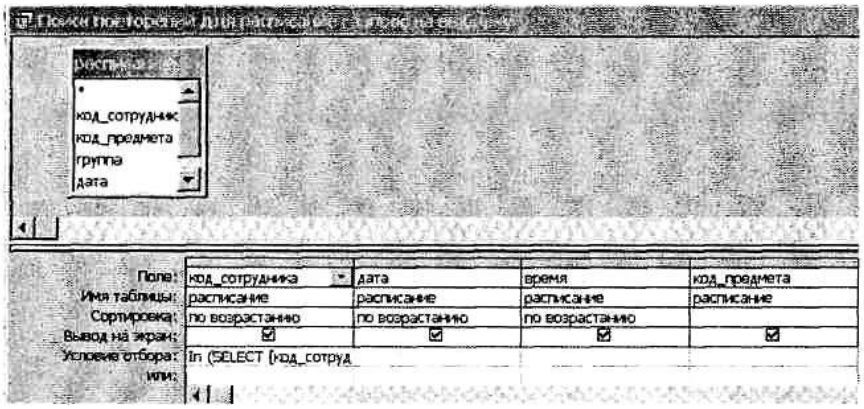


Рис. 6.55. Повторяющиеся записи.
Вид запроса в режиме **Конструктор**

Как видим, этот запрос значительно сложнее рассмотренных ранее и включает в себя вложенный запрос, написанный на языке SQL (см. главу 7).

6.2.19. Режим сводной таблицы и сводной диаграммы

Сводная таблица или сводная диаграмма может быть получена при работе с таблицей, запросом или формой. Проиллюстрируем возможности применения сводных таблиц/диаграмм, используя в качестве источника данных запрос, базирующийся на связанных таблицах.

Предположим, что требуется проанализировать выплаты сотрудникам вуза. Для этого создадим запрос, включающий нужные для анализа поля, и выберем режим **Сводная таблица** (рис. 6.56).

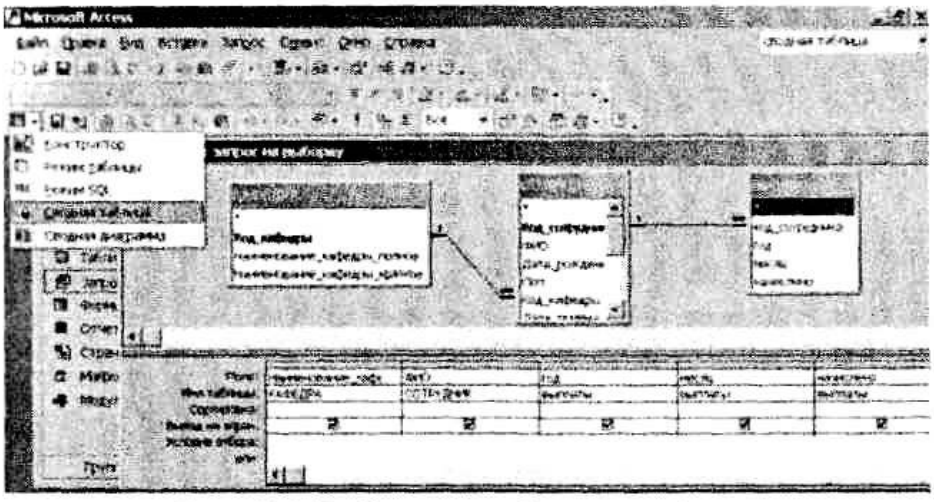


Рис. 6.56. Выбор режима **Сводная таблица**

При первом открытии запроса в режиме сводной таблицы или сводной диаграммы в эти режимы не включаются поля из источника записей. На рис. 6.57 показан вид экрана при первом открытии запроса в режиме Сводная таблица.

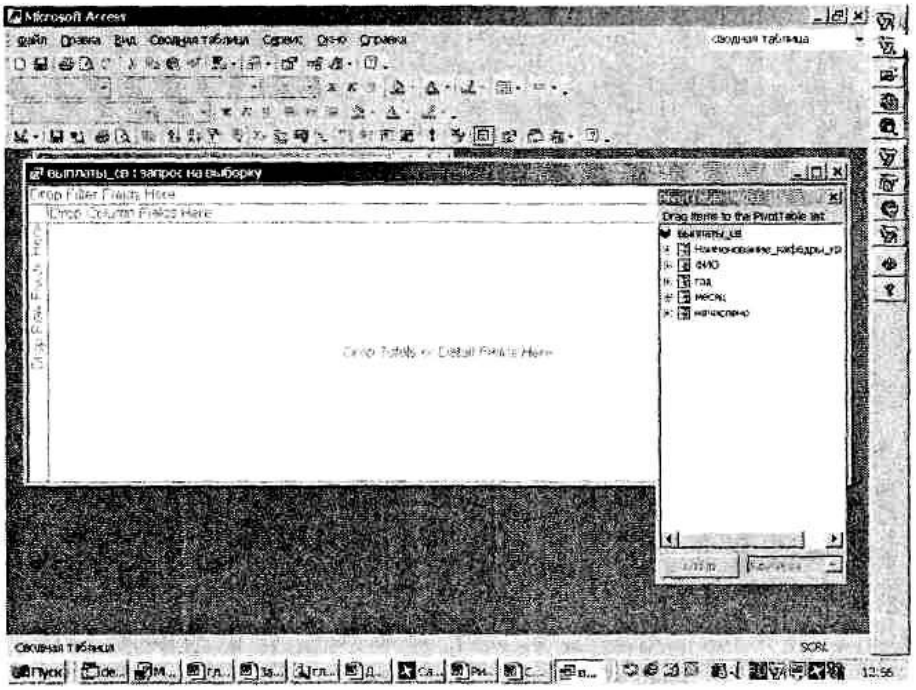


Рис. 6.57. Структура сводной таблицы (незаполненная)

После этого необходимо разместить поля по зонам сводной таблицы в соответствии с желаемым видом получаемого отчета. Для этого следует позиционироваться на соответствующем поле и в ниспадающем списке (рис.6.58) выбрать название той области, в которую должен быть помещен данный элемент.

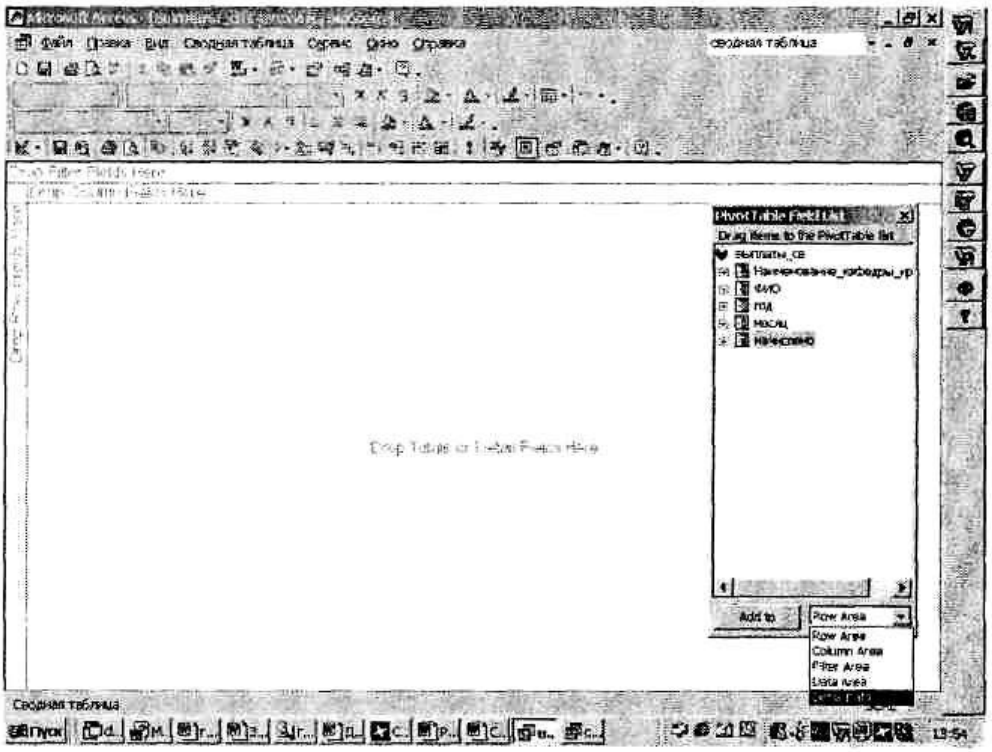


Рис. 6.58. Размещение полей по областям сводной таблицы

На рис.6.59 представлен вид спроектированной сводной таблицы.

На рис.6.60 показан вид сводной диаграммы, соответствующей данной сводной таблице. Тип выбранной диаграммы может быть легко изменен.

Использование возможностей режимов сводной таблицы и сводной диаграммы позволяет достаточно просто и быстро анализировать данные в разных разрезах и может считаться инструментом OLAP.

Наименование_кафедры_кра ▾	ФИО ▾	
All	All	
	год ▾	

	2000	2001	Grand Total
месяц ▾	начислено ▾	начислено ▾	Sum of начислено
1	1 000,00р.	1 000,00р.	8 000,00р.
	2 000,00р.	2 000,00р.	
	1 000,00р.	1 000,00р.	
	4 000,00р.	4 000,00р.	
2	10 000,00р.	10 000,00р.	112 000,00р.
	5 000,00р.	15 000,00р.	
	30 000,00р.	30 000,00р.	
	6 000,00р.	6 000,00р.	
	51 000,00р.	61 000,00р.	
3	4 000,00р.	14 000,00р.	87 400,00р.
	1 200,00р.	11 200,00р.	
	5 000,00р.	25 000,00р.	
	2 000,00р.	25 000,00р.	
	12 200,00р.	75 200,00р.	
4	2 000,00р.		4 000,00р.
	2 000,00р.		
	4 000,00р.		
5	1 000,00р.		1 000,00р.
	1 000,00р.		
6	5 550,00р.		5 550,00р.
	5 550,00р.		
7	10 000,00р.		10 000,00р.
	10 000,00р.		
8		1 000,00р.	1 000,00р.
		1 000,00р.	
9		500,00р.	500,00р.
		500,00р.	
10	2 000,00р.		3 000,00р.
	1 000,00р.		
	3 000,00р.		
11	1 000,00р.		1 000,00р.
	1 000,00р.		
12		200,00р.	200,00р.
		200,00р.	
Grand Total	91 750,00р.	141 900,00р.	233 650,00р.

Рис. 6.59. Вид спроектированной сводной таблицы

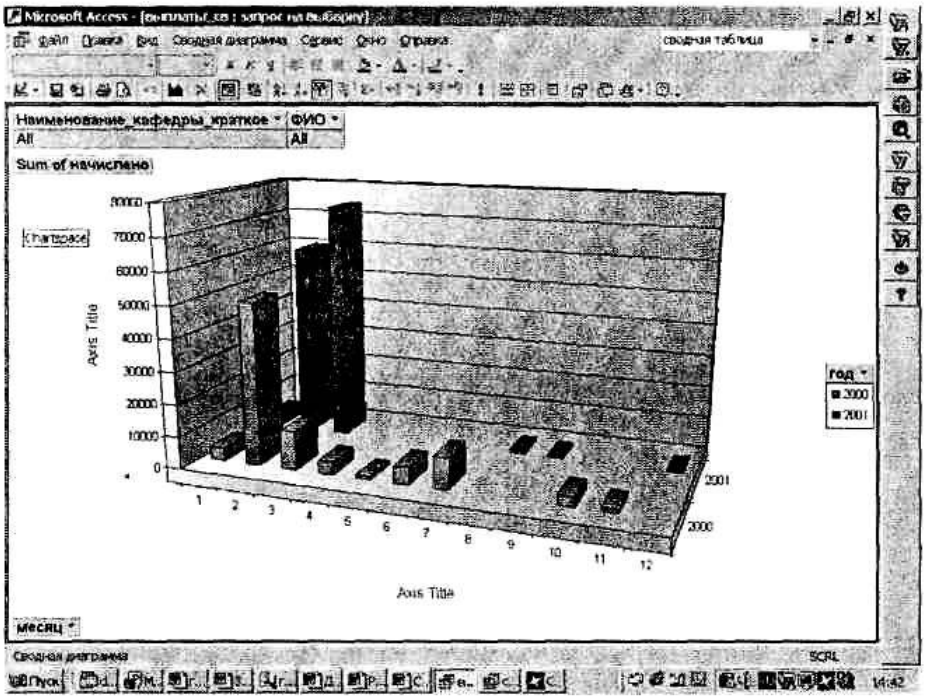


Рис. 6.60. Вид данных в режиме сводной диаграммы

На это следует обратить внимание

- Табличные языки запросов типа QBE являются простыми языками, ориентированными на конечных пользователей.
- Табличные языки запросов типа QBE реализованы во многих СУБД. Они могут отличаться некоторыми деталями реализации, но принципы, на которых они основываются, - едины.
- Не все типы запросов могут быть реализованы на QBE.
- Язык QBE часто используется в качестве построителя запросов на SQL.

Контрольные вопросы

1. К каким классам языков запросов относится язык QBE?
2. Почему QBE называется «табличным двухмерным» языком запросов?
3. Каким образом можно создавать запросы на языке QBE в Access?
4. Какие еще языки запросов можно использовать в Access?
5. Что может служить источником данных для запроса?
6. Какие разновидности запросов вы знаете?
7. Какие запросы называются «сложными»?
8. Как задаются условия «И» и «ИЛИ» в запросах?
9. В чем особенности выполнения запросов на связанных таблицах?
10. Что собой представляют перекрестные запросы? Как и для чего они создаются?
11. Что собой представляют параметрические запросы? Как и для чего они создаются?

12. Как вводятся в запрос вычисляемые поля?
13. Как можно получать итоговые значения в запросах?
14. Какие разновидности корректирующих запросов вы знаете? Как их задать?
15. Что значит «открыть запрос»?
16. Что происходит при открытии запроса на выборку?
17. Что происходит при открытии корректирующего запроса?
18. Может ли запрос на удаление не содержать в бланке запроса ни одного поля? Если да, то к каким действиям приведет выполнение этого запроса?
19. Что произойдет при открытии запроса на удаление, если в бланк запроса перенесено какое-то одно поле, но условие отбора не задано?
20. Как можно сохранить запрос? Для чего это делается?
21. Как можно сохранить результат выполнения запроса? Для чего это делается?
22. В чем разница между сохранением запроса и сохранением результата запроса?
23. Как можно задать диапазон в условии запроса?
24. Как задается состав полей, выводимых в ответ?
25. Как можно упорядочить данные в ответе?
26. Как влияет тип объединения таблиц на результат выполнения запроса?
27. Какого типа запросы можно создать с использованием Мастера «Повторяющиеся записи»?
28. В чем смысл «поиска записей, не имеющих подчиненных»?

Глава 7 ЯЗЫК SQL

7.1. Общая характеристика SQL

Одним из самых распространенных языков запросов является язык SQL (Structure Query Language). Он был разработан в середине 1970-х гг. (IBM). Первой коммерческой системой, в которой реализован этот язык, была система Oracle (1979 г.). В дальнейшем он был реализован в целом ряде популярных СУБД для различных типов ЭВМ и операционных систем. В некоторых СУБД, таких, как Oracle, INGRES, MS SQL-сервер и др., язык SQL используется в качестве основного. В других системах, например СУБД семейства dBase, Access, AD ABAS и других, этот язык применяется в качестве альтернативного.

Предшественником SQL был язык SEQUEL (Structured English Query Language).

В силу широкого использования язык SQL сначала стал де-факто стандартом языка запросов, а затем было официально разработано несколько международных стандартов SQL*. Кроме того, некоторые страны имеют национальные стандарты SQL.

Десятки фирм-разработчиков программного обеспечения создали собственные версии (диалекты) языка SQL. Несмотря на наличие стандарта, эти версии различаются как по синтаксису, так и по другим характеристикам. Есть возможности, которые реализованы во многих системах. Использование возможностей языка, реализованных в большинстве систем, упрощает создание гетерогенных систем, а также не создает дополнительных проблем в случае перевода ИС в среду другой СУБД. Однако ориентация на применение только стандартных возможностей SQL может ограничить доступ к дополнительным функциональным возможностям отдельного сервера.

Язык SQL близок к классу языков реляционного исчисления кортежей и используется в основном в реляционных СУБД. Но в связи с его очень широким применением имеются попытки реализации SQL и в других классах систем (например, в сетевой системе db_VISTA III).

Язык SQL обладает развитыми возможностями и может быть использован как конечными пользователями для формулировки не очень сложных запросов, так и специалистами в области обработки данных (прикладными программистами, администраторами баз данных). С развитием SQL наблюдается тенденция к его усложнению.

Язык баз данных SQL включает в себя два языка.- язык определения схемы (SQL-DDL) я язык манипулирования данными (SQL-DML). Язык DDL позволяет описывать и создавать такие объекты базы данных, как таблицы, индексы, представления и др. Язык DML дает возможность задавать поисковые и корректирующие запросы к базе данных. Операторы языка манипули-

* Стандарт SQL был впервые опубликован в 1986 г. и обновлялся в 1989 и 1992 гг.

рования данными SQL могут использоваться как самостоятельно (интерактивный или автономный SQL), так и совместно с операторами других языков манипулирования данными (встроенный SQL).

SQL является языком, ориентированным на групповую обработку данных. Оператор SQL возвращает информацию в табличный формат реляционной модели. При этом возникает конфликт между SQL и традиционными языками программирования, которые ориентированы на позаписную обработку данных. Это обусловило разработку встроенного SQL, который позволяет использовать SQL-операторы в кодах программ, написанных на традиционных языках программирования.

Имеется два типа встроенных SQL-операторов: статический SQL и динамический SQL. Статический SQL ссылается на SQL-операторы, которые известны до момента запуска и в дальнейшем не изменяются, в то время как динамические SQL-операторы определяются только во время выполнения программы.

Кроме DDL и DML иногда в качестве самостоятельного подмножества выделяют язык управления данными (DCL - Data Control Language). Операторы DCL обычно используются для создания объектов, относящихся к управлению доступом пользователей к базе данных, а также для назначения пользователям подходящих уровней привилегий доступа.

SQL является языком высокого уровня. Пользователь не должен при его использовании помнить об открытии и закрытии каких-либо таблиц, определять наиболее эффективный способ реализации запроса, активизировать индексы и т.п. Все это система делает автоматически. Во многих современных СУБД имеются построители запросов SQL. Обычно в этом качестве выступают языки типа QBE. Но не все типы запросов SQL могут быть реализованы на QBE. Некоторые типы запросов, например запрос-объединение (Union), невозможно создать на QBE.

7.2. Описание базы данных

7.2.1. Описание таблиц

SQL оперирует с данными, представленными в виде таблиц. Каждая таблица содержит множество строк. Различают базовые таблицы — таблицы, определенные с помощью их описания на языке описания данных, и *производные таблицы*, получаемые из одной (или нескольких других) таблицы путем выполнения некоторого запроса.

Таблица описывается с помощью оператора CREATE TABLE. Таблица определяется путем задания содержащихся в ней столбцов. Все значения в столбце имеют один и тот же тип данных. При описании таблиц могут задаваться ограничения целостности данных.

В стандарте SQL-92 было определено семь типов данных. Однако большинство из них имеют подтипы с собственными именами, используемыми

для задания типа данных при определении таблицы. В SQL-99 введено четыре новых типа данных, причем два из них -составные.

Конкретные реализации СУБД работают с различными типами данных. Причем наблюдаются случаи, когда СУБД не поддерживает некоторые из типов данных, определенных в стандарте SQL, и когда, напротив, содержит типы, которые в стандарт не включены. Используемые типы данных и форматы их представления особенно важны при обработке распределенных запросов в гетерогенной среде.

Другая важная характеристика языка - это поддержка неопределенных значений. Некоторые системы позволяют работать с неопределенными значениями, другие — нет. Эта характеристика является более значимой, чем может показаться на первый взгляд, поскольку поддержка неопределенных значений означает, кроме прочего, использование трехзначной логики при обработке значений вместо двузначной.

Структуру существующей таблицы можно изменять с помощью оператора ALTER TABLE.

Таблицы могут быть удалены с помощью оператора DROP TABLE.

Таблицы, используемые в каком-либо приложении, включаются в базу данных. База данных обычно содержит несколько таблиц, а также и цдексы и некоторые другие информационные объекты. В некоторых СУБД имеется оператор CREATE DATABASE. В стандарте SQL для аналогичных целей используется оператор CREATE SCHEMA.

Метаинформация в SQL-системах хранится также в виде реляционных таблиц (базы данных, основанные на таком принципе, называют *самоописываемыми*).

7.2.2. Ограничения целостности

Декларативное задание ограничений целостности. Ограничения целостности определяют допустимые значения базы данных пу-тем ограничения значений в базовых таблицах. В SQL большая часть ограничений целостности задается при описании таблицы. Возможность декларативного задания ограничений целостности является одним из существенных отличий стандарта SQL-89 и последующих от стандарта SQL-86.

При описании базы данных может быть задан ряд ограничений целостности.

1. Спецификация уникальности:
   - определение первичного ключа;
   - задание признака уникальности поля/совокупности полей.
2. Ограничение на столбец:
   - ограничение на неопределенное значение;
   - задание проверочных ограничений на строки таблицы (ограничения контроля).
3. Определение ограничений целостности связей (ссылки между таблицами);

К этому типу ограничений примыкает возможность задания значений столбца по умолчанию при занесении строки в таблицу.

Ограничения целостности задаются с помощью определенных конструкций языка описания схемы при описании таблицы.

Определение ограничения уникальности (unique constraint definition) требует, чтобы никакие две строки в таблице не имели одинаковых значений в указанном столбце или совокупности столбцов.

Это ограничение позволяет определить вероятные ключи таблицы. Тот из вероятных ключей, который выбран в качестве первичного, описывается как PRIMARY KEY. Ограничение PRIMARY KEY может использоваться только один раз для каждой таблицы. Остальные вероятные ключи могут быть определены путем использования ограничения UNIQUE.

Спецификация NOT NULL требует, чтобы никакие значения в столбце не были неопределенными значениями.

Для определения ограничения ссылок (referential constraint definition) используются ограничения FOREIGN KEY и REFERENCES. После ключевых слов FOREIGN KEY в круглых скобках указывается список имен описываемой таблицы (ссылающейся таблицы, • порожденной таблицы). После ключевого слова REFERENCES указывается имя таблицы, которая содержит родительский ключ, и далее в круглых скобках - список столбцов, составляющих этот первичный ключ. В принципе ссылающаяся таблица может быть той же самой таблицей, что и целевая таблица.

Определение проверочного ограничения (check constraint definition) требует, чтобы указанное в нем условие (search condition) было истинным для каждой строки таблицы.

Момент проверки ограничения целостности зависит от конкретной реализации. Стандарт SQL-92 позволяет определить ограничения таким образом, чтобы они не проверялись до завершения текущей транзакции. Вводятся понятия *неотложенное и отложенное* ограничение целостности. По умолчанию ограничения определяются как неотложенные. Неотложенные ограничения проверяются при выполнении каждого оператора SQL (SQL statement). Если базовая таблица, ассоциированная с ограничением целостности, не удовлетворяет его условию, то оператор SQL не выполняется.

Триггеры. Триггер представляет собой специальную хранимую процедуру, которая не вызывается непосредственно ни из клиентского приложения, ни из другой хранимой процедуры, а неявно активизируется при выполнении той или иной связанной с ним операции (события).

Триггеры обычно ассоциируют с операциями обновления (UPDATE), вставки (INSERT) и удаления (DELETE) в таблице. Триггеры часто используются для задания различных проверок целостности данных при выполнении упомянутых выше операций.

Триггеры не определены в стандарте SQL, но поддерживаются практически во всех SQL-серверах и даже во многих настольных СУБД. Типы возможных триггеров и их синтаксис сильно различаются от системы к системе.

7.3. Запросы на выборку

Основным оператором языка SQL, позволяющим осуществлять отбор информации из базы данных, является оператор SELECT, который в простейшем виде может быть задан следующим образом:

SELECT <список колонок, включаемых в ответ>
FROM <список таблиц>
WHERE <условие>.

Предложения SELECT (отобрать) и FROM (из) должны присутствовать обязательно. Условие WHERE (где) может быть опущено. Тогда в ответ войдут все строки, имеющиеся в таблице (если быть точным, то надо иметь в виду, что такой оператор должен соответствовать реляционной операции «проекция», и если в результате проекции на выбранные столбцы будут появляться дублирующие строки, то они должны быть удалены из ответа; однако SQL позволяет управлять выводом в ответ повторяющихся строк, и можно добиться как вывода только уникальных строк, так и включения в ответ повторяющихся строк).

Оператор SELECT может включать в себя и другие предложения, позволяющие, в частности, осуществлять упорядоченность ответа, выполнять обобщающие функции. Полный синтаксис оператора SELECT будет рассмотрен позже.

Если в ответ должны войти все колонки, имеющиеся в исходной таблице, то вместо их перечисления в SELECT можно поставить знак «*».

Так, например, запрос «Выдать всю информацию из таблицы «Kadr» по сотрудникам, чей возраст (vozr) равен 40 годам» {см. рис.6.1) может быть представлен на SQL следующим образом:

SELECT * FROM kadr **WHERE** vozr=40;

Условие, задаваемое в предложении WHERE, может быть простым и сложным. Для формулирования сложного условия могут быть использованы логические операторы AND и OR. Так, например, запрос «Выдать всю информацию из таблицы «Kadr» по мужчинам, чей возраст равен 40 годам» (т.е. пол=«*мужской*» И «*возраст=40 лет*») может быть представлен на SQL следующим образом:

SELECT * FROM kadr **WHERE** vozr = 40 **AND** pol = "м";

Оператор SELECT оперирует над множествами и результатом обработки в общем случае является множество строк. К этим множествам могут быть применены теоретико-множественные операции объединение (UNION), пересечение (INTERSECTION), разность (DIFFERENCE, MINUS, EXCEPT) и

др. В разных реализациях языка SQL наборы теоретико-множественных операций различаются.

Язык SQL позволяет запрашивать вычисляемые значения. В этом случае в предложении SELECT указывается выражение для вычисления значения колонки. Например, в приведенном ниже предложении запрашивается вывод стоимости поставленных товаров путем ее вычисления на основе хранящихся в таблице «Postypl» данных о количестве (kolv) поставленных товаров и их цены {pena):

```
SELECT naimprod, datapost, kolv*cena
FROM postypl;
```

Запрос может быть простым, состоящим из одного оператора SELECT, и вложенным, когда один оператор SELECT включается в состав другого оператора. Этот включенный оператор называется *подзапросом* (subselect) или *подчиненным запросом*. Существуют два типа вложенных подзапросов: *обычный* и *коррелированный*. В обычном подзапросе внутренний запрос выполняется первым, и его результат используется для выполнения основного запроса. В коррелированном подзапросе внешний запрос выполняется первым, и его результат используется для выполнения внутреннего запроса. Внутренний запрос выполняется для каждой строки, возвращенной внешним запросом.

В запросе можно указать упорядоченность ответа по определенному признаку (полю, совокупности полей, выражению).

Возможна подгруппировка данных в целях получения подытогов или других обобщающих величин (среднее, минимум, максимум и др.). Набор агрегатных функций отличается в разных системах. В стандарт SQL-92 включены следующие агрегатные функции: Count - подсчет, Avg - среднее, Sum - сумма, Max - максимум, Min - минимум.

В запросе допускается только один уровень группировки. Группировка может осуществляться как по одному полю, так и по совокупности полей.

При выполнении запроса может возникнуть необходимость соединения двух или более таблиц. Возможны разные способы задания условия соединения (вложенные запросы, задание условия соединения в предложении WHERE, операция JOIN в предложении FROM). Более подробно эти вопросы будут рассмотрены ниже.

В некоторых реализациях языка SQL отобранные оператором SELECT данные могут быть сохранены в виде таблицы базы данных.

7.4. Возможности корректировки хранимых данных

Хотя SQL первоначально создавался как язык запросов, в настоящее время он является комплексным языком запросов-обновлений. Кроме оператора SELECT, позволяющего проводить отбор данных, SQL включает в себя операторы корректировки БД: включение новой записи (INSERT), обновле-

ние отдельных полей (UPDATE), удаление записи или группы записей (DELETE).

Операторы языка манипулирования данными SQL могут использоваться как самостоятельно, так и совместно с операторами других языков манипулирования данными.

7.5. Создание представлений (VIEW)

Представление - это виртуальная таблица, данные для которой получаются из базовых таблиц или других представлений. Представление может быть получено из одной таблицы или нескольких; оно может включать в себя вычисляемые поля.

Представление можно рассматривать как хранимый запрос (оно и выражается с помощью запроса). Представление может использоваться в запросах наряду с реальными таблицами. Оно упрощает создание сложных запросов.

Синтаксис оператора CREATE VIEW различается в реализациях конкретных систем.

7.6. Создание и использование курсоров

Курсор похож на представление в том, что при его определении используется запрос. Когда курсор открыт, он содержит результат этого запроса. Но назначение представления и курсора различно. Курсоры используются только во встроенном SQL.

Курсор создается с помощью команды

DECLARE имя_курсора **CURSOR**
FOR подзапрос;

Курсоры используются для согласования теоретико-множественного языка SQL и включающих языков, которые являются языками позаписной обработки. Для выборки текущей записи активного множества используется оператор FETCH.

2.7. Управление транзакциями

Транзакция представляет собой группу последовательных SQL-операторов, которые переводят базу данных из одного целостного состояния в другое. В случае возникновения какого-либо сбоя (ошибки) результаты аннулируются и обеспечивается «откат» базы данных в исходное состояние.

Способ, каким осуществляется требование начать выполнение транзакции, зависит от конкретной реализации SQL. В стандарте ANSI средств для явной активизации начала работы транзакций не предусмотрено.

К операторам, позволяющим управлять транзакциями, относятся:

- COMMIT - завершает выполнение транзакции и пытается зафиксировать все изменения, вносимые транзакцией;

- ROLLBACK - используется для того, чтобы отменить все изменения, вносимые транзакцией.

В SQL-92 есть еще оператор SET TRANSACTION, который позволяет назначить транзакции имя и задать дополнительные свойства транзакции.

Конкретные СУБД иногда включают команды управления транзакциями, которые не предусмотрены в стандарте.

7.8. Стандартный SQL-92

7.8.1. Создание объектов

Виды объектов

В стандарте SQL-92 определяются следующие виды объектов:

- ***Table*** - таблица;
- ***View*** - представление;
- ***Schema*** ~ схема;
- ***Domain*** - домен;
- ***Assertion*** - утверждение;
- ***Character set*** - набор символов;
- ***Collation*** - последовательность сортировки;
- ***Translation*** - преобразование одного набора символов в другой.

Рассмотрим здесь вопросы создания таблиц и доменов. Создание представлений будет рассмотрено позже, поскольку при его определении используется оператор SELECT, который нами еще детально не изучен.

Объект ***Assertion*** используется для задания ограничений, которые должны быть проверены при работе с базой данных. Утверждение существует независимо от определения таблицы и может иметь ссылки на любую постоянную таблицу схемы. Утверждение может, например, использоваться для проверки, не является ли данная таблица пустой, так как выяснить это, используя ограничения целостности, заданные при описании таблицы, нельзя.

Определение таблицы

Для создания таблицы используется оператор CREATE TABLE, который имеет следующий синтаксис:

CREATE TABLE [{GLOBAL|LOCAL}|TEMPORARY] имя таблицы
({определение столбца|[ограничение таблицы]}.,..
[ON COMMIT{DELETE|PRESERVE} ROWS]);

Базовые таблицы бывают постоянные или временные. Если создается временная таблица, то используется параметр TEMPORARY. В этом случае при определении таблицы должны быть указаны дополнительные параметры GLOBAL или LOCAL.

Предложение ON COMMIT может использоваться только для временных таблиц. Оно определяет, что происходит со строками временных таблиц по завершении транзакции. По умолчанию подразумевается ON COMMIT DELETE ROWS.

При описании таблицы должен быть определен хотя бы один столбец и могут быть определены ограничения для таблицы. Столбец определяется следующим образом:

определение столбца ::=
имя столбца|{имя домена
|тип данных [размер]}
[ограничение столбца...]
[**DEFAULT** значение по умолчанию]
[**COLLATE** имя сравнения]

Ограничение на длину и допустимые символы, используемые в именах столбцов, зависят от реализации.

В стандарте SQL-92 принято семь типов данных, но большинство из них имеют подтипы с собственными именами, используемыми при задании типов данных для столбцов или доменов. Именно каждый подтип обычно и считается типом данных, однако для работы самой СУБД нет необходимости различать типы данных в пределах категорий.

Стандарт предусматривает следующие категории данных:

- CHARACTER STRING (CHARACTER, CHARACTER VARYING);
- NATIONAL CHARACTER (NATIONAL CHARACTER, NATIONAL CHARACTER VARYING);
- BIT STRING (BIT, BIT VARYING);
- EXACT NUMERIC (NUMERIC, DECIMAL, INTEGER, SMOLLINT;
- APPROXIMATE NUMERIC (FLOAT, REAL, DOUBLE PRESION);
- DATETIME (DATE, TIME, TIMESTAMP);
- INTERVAL.

Если вместо типа данных указывается имя домена, то, естественно, что этот домен должен быть предварительно определен.

Ограничения, указанные в описании после определения столбца, относятся к этому столбцу.

Возможны следующие ограничения:

- NOT NULL;
- PRIMARY KEY;
- UNIQUE;
- CHECK;
- FOREIGN KEY/REFERENCES.

Если для столбца задано ограничение NOT NULL, то либо должно быть определено значение по умолчанию, либо каждый оператор INSERT или UPDATE не должен приводить к появлению NULL.

Если столбец определен на домене, который имеет собственное значение по умолчанию, и для этого столбца также задается значение по умолчанию, то значение столбца переопределяет значение домена.

Ограничение PRIMARY KEY служит для задания первичного ключа. Как известно, ключ может быть простым, состоящим из одного поля, и составным, включающим несколько полей. В первом случае ограничение может быть задано или непосредственно при описании поля (см. пример описания таблицы «Spr_mat» ниже), или как ограничивающее условие в конце описания таблицы (см. пример описания таблицы «Spr_post» ниже). Если ключ составной, то может быть использован только второй способ задания ограничения, причем поля, входящие в состав ключа поля, указываются в скобках и разделяются запятыми (см. пример описания таблицы «Post» ниже). В таблице может быть задано только одно ограничение PRIMARY KEY

Ограничение UNIQUE также может относиться как к отдельному полю, так и к совокупности полей. Оно похоже на ограничение PRIMARY KEY, но в отличие от него, во-первых, может задаваться для нескольких разных полей или совокупностей полей и, во-вторых, для них разрешено значение NUL.

Внешний ключ (FOREIGN KEY) - это столбец в дочерней таблице, ссылающийся на ключ родительской таблицы. Использование внешних ключей является основным механизмом поддержания ссылочной целостности внутри реляционной базы данных. При описании внешнего ключа указывается также ссылка на соответствующие поля в родительской таблице (REFERENCES) (см. пример описания таблицы «Spr_post» ниже). В родительской таблице для столбца или совокупности столбцов, на которые идет ссылка, должно быть задано ограничение на уникальность. Если внешний ключ составной, то при описании таблиц должны быть выполнены следующие условия: для каждого из полей, на которые идет ссылка, должно быть задано ограничение NOT NULL; число столбцов в ссылке на таблицу и столбцы должно соответствовать числу столбцов во внешнем ключе; *i*-й столбец во внешнем ключе соответствует *i*-му столбцу в ссылке на таблицу и столбцы, тип данных и длина поля каждого столбца ссылки должны соответствовать типу данных соответствующего целевого столбца и иметь равную длину.

При задании ограничения CHECK в скобках указывается предикат, который использует значения столбцов в выражении для вычисления значения. Предикат может принимать значения TRUE, FALSE или UNKNOWN. Ограничение считается нарушенным, когда предикат принимает значение FALSE.

Стандарт SQL-92 позволяет определять ограничения так, чтобы они не проверялись до завершения текущей транзакции.

Ниже приведен пример описания трех связанных таблиц: «Spr_mat» — справочник материалов, «Spr_post» — справочник поставщиков и «Post» - таблица, содержащая информацию о поставках. При включении в базу данных информации о конкретной поставке должно проверяться наличие вводи-

мых значений кода материала (kod_mat) и кода поставщика (kod_post) в соответствующих справочниках.

CREATE TABLE spr_mat
(kod_mat **CHAR**(3) **NOT NULL PRIMARY KEY,**
naim_mat **CHAR** (30) **NOT NULL UNIQUE);**

CREATE TABLE spr_post
(kod_post **CHAR(5) NOT NULL**,
naim_post **CHAR (30) NOT NULL**,
PRIMARY KEY (kod_post),
UNIQUE (naim_post));

CREATE TABLE post
(kod_post **CHAR(5) NOT NULL**, kod_mat **CHAR(3) NOT NULL**,
data_post **DATE NOT NULL**, kolv **NUMERIC NOT NULL**,
PRIMARY KEY (kodjpost, kodjnat, datajpost),
FORIGN KEY (kodjnat) **REFERENCES** sprjnat (kodjnat),
FORIGN KEY (kod_post) **REFERENCES** spr_post (kod_post));

Структуру существующей таблицы можно изменить с помощью оператора

ALTER TABLE имя таблицы
{**ADD [COLUMN]** определение столбца}
| {**ALTER [COLUMN]** <имя столбца> <изменяющее действие>}
| {**DROP [COLUMN]** имя столбца **RESTRICT | CASCADE**}
| {**ADD** определение ограничения для таблицы}
| {**DROP CONSTAINT** имя ограничения **RESTRICT | CASCADE**};

изменяющее действие::=
{**SET DEFAULT** значение по умолчанию} | {**DROP DEFAULT**}

Как видно, с помощью оператора ALTER TABLE можно добавить новый столбец в таблицу, изменить определение существующего столбца, добавив/удалив значение по умолчанию, добавить/удалить ограничения целостности, а также удалить столбец из таблицы. При удалении столбца, если будет использован параметр RESTRICT, то в случае наличия каких-либо ссылок на этот столбец (в представлениях, ограничениях, условиях) оператор будет отвергнут. Если же будет использован параметр CASCADE, то все объекты, имеющие ссылки на этот столбец, будут уничтожены.

Оператор ALTER TABLE присутствует не во всех СУБД. Часто реализация этого оператора не полностью соответствует стандарту.

Определение домена

Домен позволяет определить альтернативный тип данных. Домен создается оператором CREATE DOMAIN:

CREATE DOMAIN имя домена [**AS**] тип данных
[**DEFAULT** значение по умолчанию]
[определение ограничения...]
[**COLLATE** имя сравнения];

Домен имеет смысл создавать, когда определенный с его помощью тип данных используется в создаваемой базе данных многократно. При описании таблицы для соответствующих полей вместо типа данных будет указываться имя домена.

7.8.2. Запросы

Оператор SELECT

Общая характеристика оператора. Для отбора информации из базы данных служит оператор SELECT. Синтаксис оператора выглядит следующим образом:

SELECT [DISTINCT]
{{функция агрегирования.. | выражение для вычисления значения
[**AS** имя столбца]}.,}
| {спецификатор.*}
|*
FROM {{имя таблицы [**A8**][имя корреляции].[(имя столбца.,..)]}
|{подзапрос [**AS**][имя корреляции.[имя столбца.,..]}
|соединенная таблица}.,..
[**WHERE** предикат]
[**GROUP BY** {{[имя таблицы| имя корреляции]}.] имя столбца}.,..}]
[**HAVING** предикат]
[**UNION | INTERSECT | EXCEPT}[ALL**]
[**CORRESPONDING [BY** (имя столбца.,..)]]
оператор select | **TABLE** имя таблицы | конструктор значений таблицы]
[**ORDER BY**{{столбец-результат [**ASC| DESC**]}.,..}
|{{положительное 4hoio[**ASC| DESC**]}.,..}]};

Оператор состоит из предложений SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY, которые должны быть записаны в команде именно в той последовательности, в которой они перечислены в синтаксической формуле.

Предложение SELECT. Оно определяет столбцы таблицы, получаемой в результате выполнения запроса. Столбец результатной таблицы может быть задан именем столбца исходной таблицы. Если в запросе используется несколько таблиц и в них имеются поля, имеющие одинаковые имена, то для указания такого поля используется конструкция <имя таблицы>.<имя поля>. Кроме того, в предложении SELECT могут использоваться любые допустимые выражения, которые зададут формулу для определения вычисляемого поля. С помощью [AS <имя столбца>] можно задать имя столбца-результата. Эту конструкцию можно использовать не только тогда, когда определяются вычисляемые поля, но и во всех других случаях, когда нужно задать имя столбца-результата, отличающееся от имени столбца исходной таблицы.

Результат выборки может в принципе содержать повторяющиеся строки. Чтобы избежать вывода повторяющихся строк в ответе, используется параметр DISTINCT.

Запросы могут использовать функции агрегирования. Стандарт предусматривает использование следующих функций агрегирования:

- COUNT - подсчет;
- SUM - сумма;
- MAX -максимум;
- MIN - минимум;
- AVG - среднее.

Чаще всего функции агрегирования используются совместно с предложением GROUP BY, но могут применяться и самостоятельно. В последнем случае результат относится не к какой-то группе, а ко всей выборке.

Существуют два типа функции COUNT. Первый тип использует символ «*». В этом случае функция подсчитывает количество строк в группе. Отдельные значения столбцов при этом не учитываются, и результат не будет зависеть от того, имеются ли в полях значения Null и указан ли параметр DISTINCT. Второй тип функции COUNT игнорирует значения Null.

Если в ответ требуется включить все поля таблицы, то для этого можно использовать символ «*». Если запрос многотабличный, то следует применять конструкцию {спецификатор.*}

Предложение FROM. В нем указываются таблицы, которые используются при формулировании запроса. Кроме этого, в качестве источника данных в запросе могут быть заданы представления.

Начиная со стандарта SQL-92 в предложение FROM можно включать встроенный оператор JOIN, который служит для задания разнообразных условий соединения таблиц, участвующих в запросе.

Типы соединения и выполняемые ими действия приведены ниже.

В качестве примера для иллюстрации различных типов соединений рассмотрена условная софтверная фирма X, которая продает программные про-

дукты. Причем это могут быть разработки как сотрудников фирмы, так и сторонних авторов.

Тип соединения	Выполняемое действие
Cross (перекрестное)	Прямое декартово произведение
Natural (естественное)	Соединение внешнего ключа со связанным с ним ключом (одноименные столбцы)
Inner (включающее или внутреннее)	Эквисоединение таблиц A и B (равные значения соответствующих столбцов)
Left (левое (внешнее))	Все строки таблицы A, а также значения из тех строк таблицы B, которые имеют совпадающие значения в поле связи
Right (правое (внешнее))	Все строки таблицы B, а также значения из тех строк таблицы A, которые имеют совпадающие значения в поле связи
Full (полное)	Объединяет левое и правое соединения
Union (соединение типа объединение)	Противоположно Inner

Таблица «А_сотрудники» содержит сведения о сотрудниках фирмы Х:

Таб_ном	ФИО
01	Диго
02	Афанасьев
03	Сидоров

Таблица «Б_разработки» содержит информацию о том, какие ПП предлагаются для распространения на фирме Х и кто является автором каждой разработки. Поскольку для сторонних разработчиков нет табельного номера (поле «А_сотрудники.таб_ном»), то в таблице «Б_разработки» фиксируются ФИО разработчиков.

ФИО	Продукт
Диго	П1
Диго	П2
Афа-	П3
Чис-	П4

Достаточно трудно определить предметную область, чтобы пример был не громоздкий и все теоретически возможные соединения имели реальную интерпретацию, поэтому некоторые примеры могут показаться несколько надуманными.

Если в предложении FROM перечислено несколько таблиц, то все они неявно считаются соединяемыми. Если тип соединения явно не задан, то счи-

тается, что каждая строка первой таблицы соединяется с каждой строкой второй таблицы. Именно такое соединение и называется перекрестным.

Результат перекрестного соединения для приведенных выше таблиц представлен ниже.

Таб_ном	А_сотрудники.фио	Б_разработки.фио	Продукт
01	Диго	Диго	П1
01	Диго	Диго	П2
01	Диго	Афанасьев	ПЗ
01	Диго	Чистов	П4
02	Афанасьев	Диго	П1
02	Афанасьев	Диго	П2
02	Афанасьев	Афанасьев	ПЗ
02	Афанасьев	Чистов	П4
03	Сидоров	Диго	Ш
03	Сидоров	Диго	П2
03	Сидоров	Афанасьев	ПЗ
03	Сидоров	Чистов	П4

Запрос на SQL может иметь следующий вид:

SELECT а_сотрудники.таб_ном, а_сотрудники.фио, б_разработки.
фио, б_разработки.продукт
FROM а_сотрудники, б_разработки;

Чаще всего при создании запросов используется тип соединения INNER JOIN, при котором соединенная таблица будет включать только те строки, для которых есть соответствующие друг другу значения полей связи в обеих таблицах.

Результат соединения типа INNER JOIN для приведенных выше таблиц представлен ниже.

Таб_ном	ФИО	Продукт
01	Диго	П2
01	Диго	П1
02	Афанасьев	ПЗ

Этот запрос показывает разработки, выполненные сотрудниками фирмы X.

На SQL такой запрос будет иметь следующий вид:

```
SELECT а_сотрудники.таб_ном, а_сотрудники.фио, б_разработки.продукт
FROM а_сотрудники INNER JOIN б_разработки
ON а_сотрудники.фио = б_разработки.фио;
```

При использовании соединения типа LEFT JOIN в результатную таблицу попадают все записи из первой таблицы и только те записи из второй таблицы, для которых есть соответствующие значения полей связи в первой таблице. Соединение типа LEFT JOIN для рассматриваемого примера даст в результате список всех сотрудников фирмы X с указанием их разработок:

Таб_ном	ФИО	Продукт
01	Диго	П2
01	Диго	Ш
02	Афанасьев	П3
03	Сидоров	Null

На SQL такой запрос будет выглядеть следующим образом:

```
SELECT а_сотрудники.таб_ном, а_сотрудники.фио, б_разработки.продукт
FROM а_сотрудники LEFT JOIN б_разработки
ON а_сотрудники.фио = б_разработки.фио;
```

При использовании соединения типа RIGHT JOIN, напротив, в результатную таблицу попадают все записи из второй таблицы и только те записи из первой таблицы, для которых есть соответствующие значения полей связи во второй таблице. Соединение типа LEFT JOIN для рассматриваемого примера даст в результате список всех продуктов с указанием разработчика и его табельного номера:

Таб_ном	ФИО	Продукт
01	Диго	Ш
01	Диго	П2
02	Афанасьев	П3
	Чистов	П4

На SQL такой запрос будет выглядеть следующим образом:

```
SELECT а_сотрудники.таб_ном, б_разработки.фио, б_разработки.продукт
```

FROM а_сотрудники RIGHT JOIN б_разработки
ON а_сотрудники.фио = б_разработки.фио;

Во всех приведенных выше примерах предполагалось, что условием соединения является равенство значений полей связи. Обычно именно этот тип сравнения и используется.

Как уже указывалось, все возможности стандартного SQL в полном объеме практически не реализованы ни в одной СУБД. Операторы INNER, LEFT, RIGHT JOIN присутствуют во многих системах, чего нельзя сказать о FULL и UNION JOIN.

FULL JOIN для нашего примера даст следующий результат:

Таб_ном	ФИО	Продукт
01	Диго	П1
01	Диго	П2
02	Афанасьев	П3
03	Сидоров	Null
Null	Чистов	П4

Результатом UNION JOIN будет

Таб_ном	ФИО	Продукт
03	Сидоров	Null
Null	Чистов	П4

Предложение WHERE. В нем задается условие отбора записей. Предикат может включать одно выражение или несколько. Части сложного условия соединяются логическими операторами AND (И) или OR (ИЛИ).

В выражениях могут использоваться следующие операторы сравнения: = (равно), о (не равно), < (меньше), <= (меньше или равно), > (больше), >= (больше или равно), которые могут предваряться оператором NOT.

Предикат может принимать одно из трех значений: TRUE, FALSE, UNKNOWN. В результатную таблицу переносятся те строки, для которых значение предиката равно TRUE.

Кроме стандартных операторов сравнения в SQL можно использовать специальные операторы предикатов:

- <интервальный предикат >;
- <предикат Ш>;
- < предикат проверки на неопределенное значение >;
- <предикат подобия>.

При использовании интервального предиката диапазон значений можно задавать в виде

WHERE [NOT]<выражение> **BETWEEN** <нижнее выражение> **AND** <верхнее выражение>.

Например, если требуется выдать сведения о поставке продукции за последнюю декаду ноября 2002 г., то запрос можно задать следующим образом:

SELECT * FROM post **WHERE** postdate **BETWEEN** #11/20/02# **AND** #11/30/02#;

Это же условие отбора можно задать и без использования интервального предиката:

postdate>=#11/20/02# **AND** postdate<= #11/30/02#;

При использовании предиката IN предложение WHERE будет иметь следующий вид:

WHERE [NOT]< выражение > **[NOT] IN** (<список значений>/<лод-запрос>).

В приведенном ниже примере требуется вывести данные о поставках поставщиков PI, P2, P3.

SELECT * FROM post WHERE kod_post IN ("p1", "p2", "p3")

Без использования IN запрос имел бы следующий вид:

SELECT * FROM post **WHERE** kod_post ="p1" **OR** kod_post ="p2" **OR** kod_post = "p3";

Пример с использованием подзапроса будет рассмотрен позже, при демонстрации возможностей обработки нескольких таблиц.

Предикат подобия применяется для поиска подстроки в указанной строке. Предложение WHERE при использовании предиката этого типа будет иметь следующий вид:

WHERE [NOT] <выражение_для_вычисления_значения_строки_1>
[NOT] LIKE <выражение_для_вычисления_значения_строки_2>.

В качестве <выражения_для_вычисления_значения_строки_1> обычно используется <имя колонки>.

<Выражение_для_вычисления_значения_строки_2> называется образцом. В образце разрешается применять заполнители (трафаретные символы):

- символ подчеркивания (_) - используется вместо любого единичного символа в проверяемом значении;
- символ процента (%) - заменяет набор любых символов в проверяемом значении.

Предположим, что коды металлов начинаются с буквы «м». Тогда запрос, позволяющий вывести сведения о поставке металлов, будет иметь вид

SELECT * FROM post **WHERE** kod_mat **LIKE** "м%";

Предикат проверки на неопределенное значение имеет вид

предикат **NULL** ::=
конструктор значения строки **IS [NOT] NULL**

Например, если в таблице «Сотрудник» (sotr) есть поле «Ученая_степень» (ych_st), то запрос, выводящий список сотрудников, не имеющих ученых степеней, будет выглядеть следующим образом:

SELECT fio **FROM** sotr **WHERE** ych_st **IS NULL**;

При использовании подзапросов в условии WHERE может быть использован квантор существования EXISTS. Формат условия WHERE в этом случае имеет вид

WHERE [NOT] EXISTS
(<подзапрос>).

EXISTS проверяет, вернул ли подзапрос какие-либо ряды. Фактически любой подзапрос, который может быть выражен с использованием IN, может альтернативным образом быть сформулирован также с использованием EXISTS. Обратное утверждение несправедливо.

Пример запроса с использованием EXISTS:

```
SELECT naimmat
FROM cennik
WHERE EXISTS
(SELECT *
FROM post
WHERE kodpost='P01' AND
cennik.kodmat=post.kodmat);
```

Предложение GROUP BY. Оно используется для определения групп выходных строк, к которым могут применяться те или иные агрегатные функции. Предложение GROUP BY всегда используется со встроенными агрегатными функциями. Обратное утверждение неверно. Агрегатные функции могут использоваться в предложениях SELECT, HAVING. Если агрегатные функции используются без предложения GROUP BY, то они будут применяться ко всему набору строк, удовлетворяющему условию запроса.

Конструкция GROUP BY работает только на одном уровне. Нельзя разбить каждую из этих групп на группы более низкого уровня, а затем применять стандартную функцию на каждом уровне подчиненности.

Например, в таблице «Zarpl», содержащей сведения о заработной плате рабочих, имеются колонки FIO (фамилия, инициалы), «ТаЬпош» (табельный номер), «Uch» (участок), «Zpl» (заработная плата). Требуется определить среднюю заработную плату по каждому участку:

```
SELECT uch, AVG(zpI)
FROM zarple
GROUP BY uch;
```

В данном примере рассматривается группировка по одной колонке. В принципе можно группировать строки таблицы по любой комбинации ее колонок. В этом случае имена колонок в предложении GROUP BY перечисляются через запятую.

Фраза GROUP BY означает логическую перекомпоновку (группировку) таблицы по указанной колонке (колонкам). Физически таблицы в базе данных не перекомпоновываются. Логика выполнения запроса при использовании GROUP BY несколько отличается от реализации обычного запроса. Фраза SELECT при использовании GROUP BY применяется к каждой группе, а не к каждой строке, как обычно.

Каждое выражение во фразе SELECT должно принимать единственное значение для группы, т.е. оно может быть либо самой колонкой, либо арифметическим выражением, включающим эту колонку, либо агрегатной функцией, которая получает в результате единственное значение для группы. Кроме того, в SELECT может быть включена константа.

Предложение HAVING. Вместе с GROUP BY может использоваться фраза HAVING, которая для групп имеет то же значение, что и фраза WHERE - для строк.

Например, запрос на выдачу списка кодов тех материалов, по которым было выполнено более чем по одной поставке, будет выглядеть следующим образом:

```
SELECT codmat
FROM post
GROOUP BY codmat
HAVING COUNT(*)>1;
```

Выражение во фразе HAVING должно принимать единственное значение для группы. Формат COUNT(*) означает подсчет всех строк таблицы.

Предложение ORDER BY. Информация, получаемая в результате реализации запроса, может быть упорядочена. Для этого используется фраза ORDER BY. Строки сортируются в соответствии со значениями столбцов, указанных в списке. В этом списке могут задаваться либо имена колонок, ли-

бо целое число, которое соответствует номеру колонки в таблице, считая слева направо. Когда колонки, по которым осуществляется упорядочение, вычисляемые или являются результатом операции UNION, то «целое» должно использоваться вместо имени колонки.

Параметр ASC/DESC означает вид сортировки (по возрастанию /по убыванию соответственно). По умолчанию принимается значение ASC.

Если в ORDER BY специфицируется список полей, то это означает упорядочение по составному ключу. Старшинство полей сортировки будет определяться порядком следования полей в списке.

Колонки, специфицированные в ORDER BY, должны быть включены в SELECT.

Запросы, затрагивающие несколько таблиц

В SQL запросы, затрагивающие несколько таблиц, могут реализовываться различными способами. Во-первых, условия соединения таблиц могут быть заданы в предложении WHERE. Во-вторых, возможно применение вложенных запросов. В-третьих, можно использовать встроенные операторы JOIN, включаемые в предложение FROM (что рассматривалось выше). Последняя из указанных возможностей была включена в SQL сравнительно недавно (начиная с SQL-92). Использование встроенных операторов JOIN часто помогает упростить написание запроса. Встроенные операторы JOIN реализованы не во всех диалектах SQL. В случае их отсутствия (или в силу привычки) можно запросы реализовать и без использования JOIN.

Один и тот же запрос может быть реализован несколькими способами.

Рассмотрим следующий пример. Пусть имеются две таблицы. Первая («Postavka») содержит сведения о поставках продукции и включает колонки: Код поставщика (*kod_post*), Код продукции {*kod_prod*), Дата поставки (*dat_post*), Количество поставленной продукции (*kolv*). Вторая таблица («Sp_post») содержит колонки Код поставщика и Наименование поставщика (*naim_pst*). Требуется выдать наименования поставщиков, которые поставляют товар с кодом P2.

Возможны следующие варианты задания этого запроса:

1- пример простого вложенного подзапроса

```
SELECT naim_post
FROM sp_post
WHERE kod_post IN
(SELECT kod_post FROM postavka WHERE kod_prod="P2");
```

2- пример коррелированного подзапроса

```
SELECT naim_post
FROM sp_post
WHERE "P2" IN
```

(**SELECT** kod__prod **FROM** postavka
WHERE sp_post.kod_post=postavka.kod_post);

3- вместо вложенных подзапросов используется задание условия связывания таблиц

SELECT naim_post
FROM sp_post,postavka
WHERE postavka.kod_post=sp_post .kod_post
AND postavka.kod_prod="P2";

4- возможность использования квантора существования при задании запроса

SELECT naim_post
FROM sp_post
WHERE EXISTS (SELECT * FROM postavka
WHERE kod_post=sp_post.kod_post
AND kod_prod="P2");

5 - использование встроенного оператора JOIN

SELECT naim_post
FROM sp_post **INNER JOIN** postavka **ON** sp_post.kod_post= postavka.kod_post
WHERE kod_prod="P2";

Следует обратить внимание, что оптимизатор должен выбирать оптимальный план реализации запросов, и в идеале время выполнения правильно заданного запроса не должно зависеть от способа его написания. Однако в действительности дело обстоит иначе, и по-разному записанные запросы, в результате выполнения которых получается один и тот же ответ, могут потребовать разного количества времени для своей реализации.

Корректирующие операторы

Оператор INSERT. Он позволяет включить в таблицу новые строки

INSERT INTO имя таблицы
[(имя столбца .,..)]
выражение запроса | конструктор значений таблицы
|{**DEFAULT VALUES**}

Если список колонок не задан, то значения должны вводиться в каждую колонку таблицы; если список колонок задан, то значения соответственно

должны вводиться в те колонки, которые перечислены в списке, и в том порядке, в котором они расположены в нем.

Элементы в списке значений могут быть константами, функциями, переменными памяти. Если эти элементы являются константами, то при их задании используются определенные разделители в зависимости от типа вводимых данных: символьные данные заключаются в кавычки, даты - в фигурные скобки, логические - в точки, числовые данные вводятся без разделителей.

Пример использования оператора INSERT:

INSERT INTO cennik
VALUES ("железо", "MOOOl", "T", 33.50);

В данном примере значения вводятся во все колонки таблицы {см. описание таблицы «Cennik»), поэтому <список колонок> не указан.

Если значения, которые необходимо ввести, являются результатом выполнения запроса, то эти значения также помещаются в специфицированные колонки и должны соответствовать им по типу. При использовании <подзапроса> в указанную таблицу вводятся данные, отобранные из другой таблицы (или даже нескольких таблиц).

Оператор UPDATE. Командой, позволяющей корректировать содержание таблицы, является оператор UPDATE, имеющий следующий формат:

UPDATE <имя таблицы> **SET** <имя столбца>=<новое значение>
[,<имя столбца>=<новое значение:»...]
[<предложение **WHERE**>];

Используя оператор UPDATE, можно изменить значения указанной колонки для всех записей таблицы, если предложение WHERE не задано, или для записей, удовлетворяющих условию запроса, если используется предложение WHERE.

Оператор

UPDATE vrbt **SET** rascen=rasscen*10;

увеличивает расценки для всех записей в таблице «Vrbt» в 10 раз.

Оператор

UPDATE vrbt **SET** vrbt=1 **WHERE** koddet="B11";

устанавливает для записей, удовлетворяющих условию KODDET = "B 11", значение поля vrbt, равное 1.

Оператор DELETE. Его можно использовать для удаления строк таблицы:

DELETE
FROM <имя таблицы>
[<предложение **WHERE**>];

Например, оператор

DELETE
FROM cennik
WHERE naimmat="железо";

удалит из таблицы «Cennik» строку с naimmat=”железо”.

Следует быть осторожным при использовании оператора DELETE, поскольку, если фраза WHERE в операторе DELETE отсутствует, будут удалены все строки таблицы. То же самое произойдет, если неправильно указать условие отбора и в результате не будет отобрано ни одной строки в таблице. Оператор DELETE физически удаляет строки таблицы.

7.8.3. Создание представлений (VIEW)

Оператор CREATE VIEW

Оператор для создания представления выглядит следующим образом (синтаксис SQL-92)*:

CREATE VIEW <имя представления> [(<список столбцов выборки>)]
AS < **SELECT** оператор>
[**WITH [CASCADED** |**LOCAL] CHECK OPTION**]);

Данный оператор создает виртуальную таблицу, состав которой определяется оператором SELECT.

Список имен столбцов должен быть обязательно определен лишь в следующих случаях:

- если хотя бы один из столбцов подзапроса не имеет имени (создается с помощью выражения, SQL-функции или константы);
- если два (или более) столбца подзапроса имеют одно и то же имя.

Если список отсутствует, то представление наследует имена столбцов из подзапроса.

Представления не хранятся в памяти в виде физических таблиц. Когда используется оператор CREATE VIEW, подзапрос, следующий за AS, не исполняется. Просто в каталоге сохраняется соответствующее описание.

---

* Многие как настольные (например, dBase, FoxPro), так и корпоративные (например, MS SQL-сервер, Oracle) СУБД поддерживают механизм представлений. СУБД Access, на примере которой рассматривались выше основные конструкции SQL, такой механизм не поддерживает. Однако аналогичного эффекта можно достичь, используя предварительно сохраненные запросы при формировании новых запросов.

Пользователь может работать в дальнейшем с представлениями как с обычными таблицами (он может даже не знать, что работает с представлением). При обращении к представлению с запросом последний преобразуется системой с учетом описания представления, и такой объединенный запрос исполняется.

Следует иметь в виду, что запрос к виртуальной таблице может оказаться невыполнимым, поскольку при его преобразовании получается некорректный оператор. Является ли заданная операция выполнимой - зависит от того, как реализован алгоритм преобразования запросов в конкретной системе.

Как видно из описания формата оператора CREATE VIEW, столбцы виртуальной таблицы могут быть заданы явно, а могут быть и опущены. В последнем случае представление будет содержать те столбцы, которые определены в подзапросе. В явном виде необходимо задавать имена столбцов в случае, если какой-нибудь столбец представления получен путем вычисления арифметического выражения либо является результатом стандартной функции или константой, т.е. в том случае, когда столбец не имеет имени и не может его наследовать. В явном виде следует задавать имена столбцов также в том случае, если без этого два (или более) столбца имели бы одинаковое имя.

Представления могут быть изменяемыми и неизменяемыми.

Уничтожить представление можно командой

DROP VIEW < имя представлениям>

Цели использования представлений

Представления можно использовать для достижения ряда целей.

1. Освобождение пользователя от просмотра не относящейся к нему информации. Причем это может быть как вертикальное усечение таблицы, когда в представление включаются не все поля, а только те, которые интересуют пользователя, так и горизонтальное - когда отбираются определенные записи. Например, пусть в отделе снабжения функции сотрудников закреплены в соответствии с группами материалов. Предположим, что коды материалов группы «металлы» начинаются с буквы М. Тогда можно определить представление, в которое входят только те строки, для которых код материала начинается с буквы М. Сотрудник, отвечающий за соответствующую группу материалов, будет работать с ним так, как если бы таблица содержала сведения только о металлах.

Допустим, что имеется базовая таблица «Post», в которой хранятся сведения о поставках всех материалов. Код материала содержится в столбце Kodmat. Тогда представление, отображающее поставку металлов, будет иметь следующий вид:

CREATE VIEW postmet
AS SELECT *

```
FROM post
WHERE kodmat LIKE "M%";
```

2. Улучшение защиты данных: пользователь видит (и может с ними работать) только часть данных, определенных для него представлением. Разработчики могут ограничить для конкретных пользователей доступ к отдельным столбцам или строкам таблицы, а также задать для представления допустимые для него привилегии.

3. Упрощение сложных запросов. Рассмотрим следующий пример. Пусть у нас имеются три таблицы. Одна из них («Postavka») содержит сведения о поставках продукции и включает колонки: Код поставщика (*kod_post*), Код материала (*kod_mat*), Дата поставки (*dat_post*), Количество поставленной продукции (*kolv*), Цена (*cena*). Вторая таблица («Sp_post») содержит колонки Код поставщика (*kod_post*) и Наименование поставщика (*naim_pst*). Третья таблица («Sp_mat») содержит Код (kod_mat) и Наименование (naim_mat) материала. В ответах на запросы и в самих запросах обычно используются не коды, а названия соответствующих сущностей. В этом случае можно создать следующее представление:

```
CREATE VIEW postr
AS SELECT naim_post, naim_mat, dat_post, kolv
FROM sp_post, postavka, sp_mat
WHERE postavka.kod_post=sp_post.kod_post
AND sp_mat.kod_mat=postavka.kod_mat;
```

Запрос на выдачу информации о поставке стали может выглядеть в этом случае следующим образом:

```
SELECT *
FROM postr
WHERE naim_mat="сталь";
```

Если не использовать представление, то связи между таблицами должны задаваться в каждом запросе, а не один раз при создании представления.

4. Предоставление пользователю дополнительной информации, не содержащейся в базовых таблицах. Так, в приведенном ниже примере в представление, введено вычисляемое поле, отражающее стоимость поставленной продукции:

```
CREATE VIEW post2 (kod_post, dat_post, kolv,summa)
AS SELECT kod_post, dat_post, kolv, kolv*cena
FROM postavka;
```

Практически цели 1 и 4 являются частными случаями цели 3.

При определении VIEW можно задать любой правильный оператор SELECT за некоторыми исключениями. Поскольку синтаксис как команды SELECT, так и команды CREATE VIEW несколько различается в разных СУБД, то затруднительно дать эти ограничения в общем виде. Поэтому при создании представлений следует внимательно посмотреть, какие ограничения накладывает конкретная реализация'языка.

Фраза **WITH CHECK OPTION** (с проверкой) указывает на то, что при корректировке содержимого таблиц должна осуществляться проверка на соблюдение заданного условия.

Ограничения при использовании представлений

К представлениям могут быть обращены не только поисковые, но и корректирующие запросы. В последнем случае операции, применяемые к представлению, должны быть отражены в базовых таблицах. Поскольку представления в принципе могут быть созданы с использованием разнообразных операторов SELECT (например, обеспечивающих соединение нескольких таблиц или использующих групповые операции или иные вычисляемые атрибуты), то это может вызвать определенные проблемы при выполнении обновления. Таким образом, не все представления являются обновляемыми (изменяемыми).

Чтобы не была нарушена целостность БД, должны соблюдаться следующие ограничения:

- нельзя корректировать данные представлений, полученных на основе нескольких базовых таблиц (в некоторых СУБД это ограничение менее жесткое);
- при определении представления не должно использоваться со единение таблицы «сама с собой»;
- нельзя манипулировать данными представлений, полученных на основе оператора группирования (GROUP BY);
- нельзя корректировать вычисляемые поля;
- представление должно содержать один и только один запрос, т.е. не допускается использование UNION, EXCEPT, INTERSECT;
- включающий запрос представления не может содержать предложение DISTINCT;
- ни на один столбец нельзя ссылаться в SELECT более одного раза.

При описании представления в качестве исходных для него могут использоваться не только базовые таблицы, но и другие представления. Представление, базирующееся на другом представлении, будет обновляемым, если исходное является обновляемым, и для вновь создаваемого представления будут соблюдены указанные выше условия.

Все эти ограничения вызваны тем, что в противном случае трудно (а иногда и невозможно) обеспечить целостность данных в БД, поскольку неяс-

но, как отобразить операцию корректировки в соответствующие базовые таблицы.

В некоторых программных продуктах не обеспечивается контроль всех перечисленных выше ограничений, поэтому при использовании представлений следует соблюдать осторожность.

Специфицировать WITH CHECK OPTION в операторе CREATE VIEW можно только для обновляемых представлений.

В SQL-99 существенно расширен диапазон представлений, над которыми напрямую могут выполняться операции обновления.

Создание представлений с использованием ERWin

Создавать представления можно и при проектировании БД с использованием CASE-систем.

В ERWin создавать представления можно на физическом уровне моделирования. Для создания представления следует воспользоваться кнопкой панели инструментов. Представление изображается в модели в виде четырехугольника с закругленными углами, ограниченного пунктирной линией. После этого нужно установить связь представления с таблицами-источниками, воспользовавшись для этого кнопкой. Для задания связи необходимо сначала щелкнуть по исходной таблице, а потом - по значку представления. При этом все поля исходной таблицы переносятся в создаваемое представление.

Для того чтобы исключить некоторые из полей или провести другие изменения в представлении, следует воспользоваться редактором представлений (View Editor).

Поскольку в целевой СУБД представление реализуется в виде оператора SQL, а, как известно, диалекты SQL различаются в разных СУБД, то и окна редактора представлений несколько различаются в зависимости от выбранной целевой СУБД. Для сравнения на рис. 7.1 представлен вид окна для целевой СУБД Access, а на рис. 7.2 - для целевой СУБД Oracle.

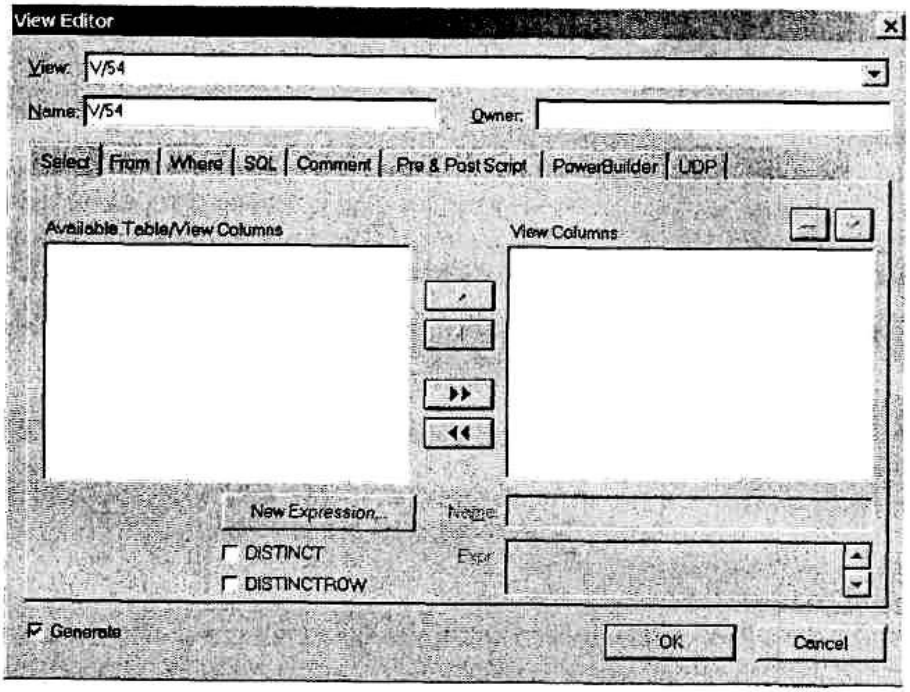


Рис. 7.1. Вид окна редактора представлений (целевая СУБД – Access)

Для того чтобы отобрать поля, которые должны входить в представление, следует воспользоваться вкладкой Select в окне редактора представлений (см. рис. 7.2).

Таблицы, на основе которых формируется представление, указываются на вкладке From (рис. 7.3). Если на этой вкладке добавить таблицы в окно From, то на схеме появится связь представления с соответствующей таблицей.

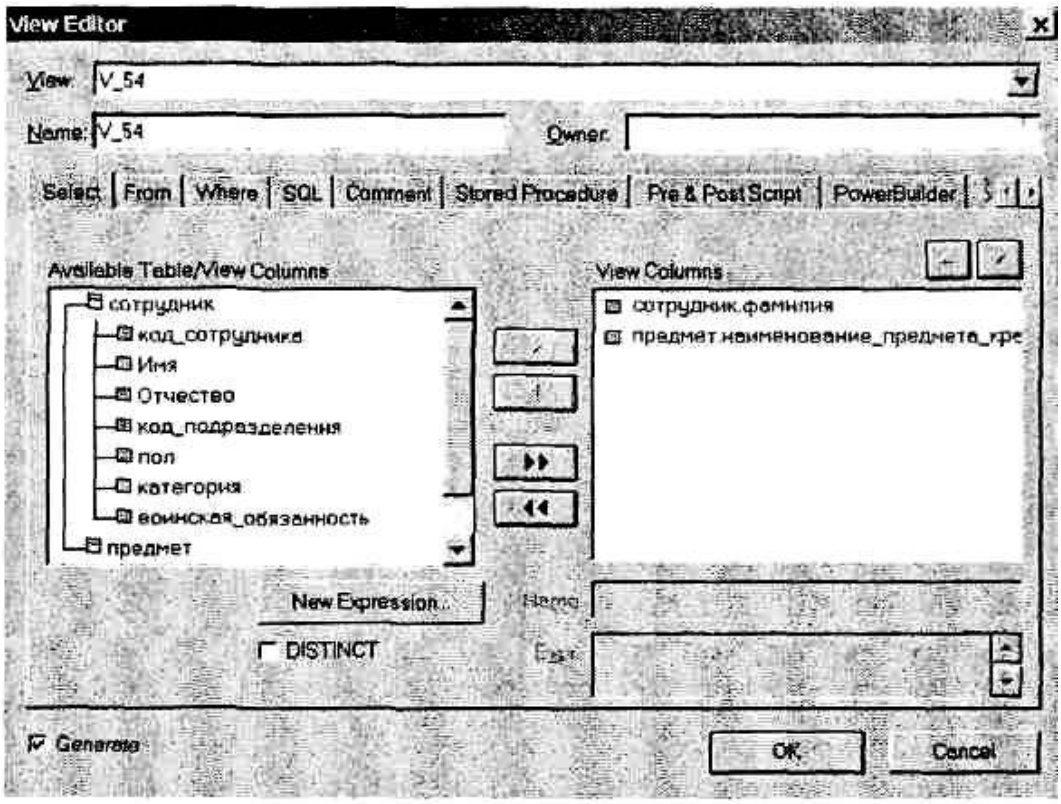


Рис. 7.2. Вид окна редактора представлений (целевая СУБД – Oracle). Отбор полей

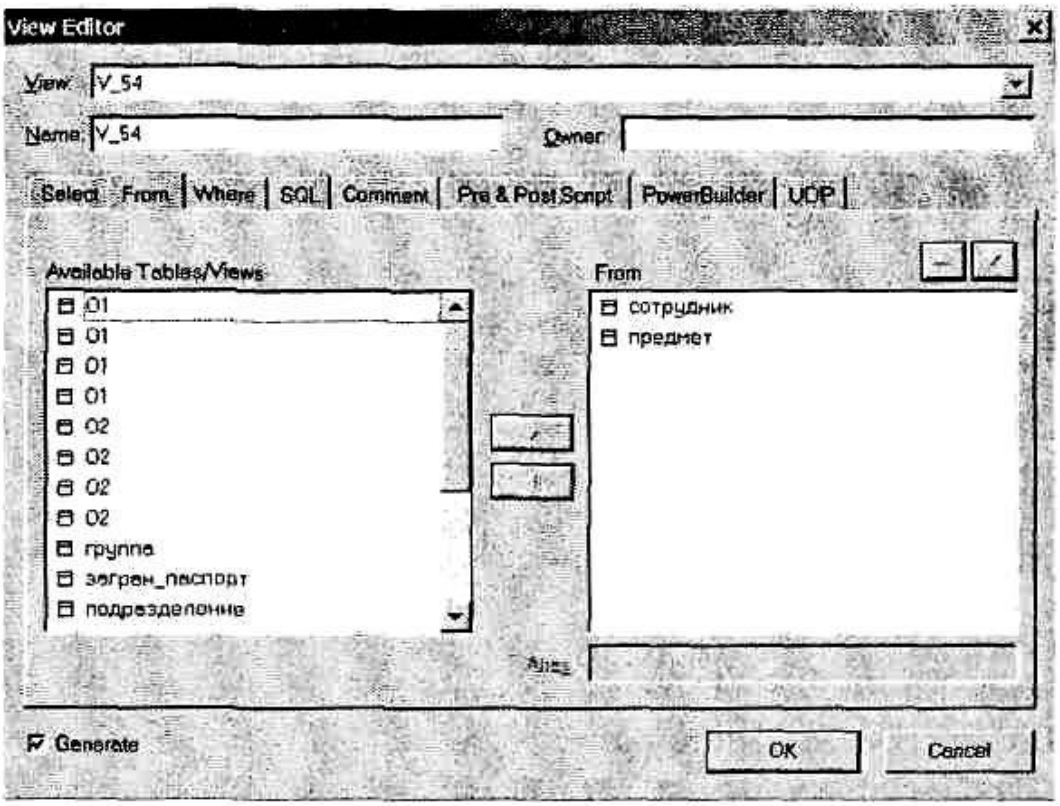


Рис. 7.3. Вид окна редактора представлений. Определение исходных таблиц

На вкладке Where (рис. 7.4) можно задать условие отбора и другие предложения оператора SELECT.

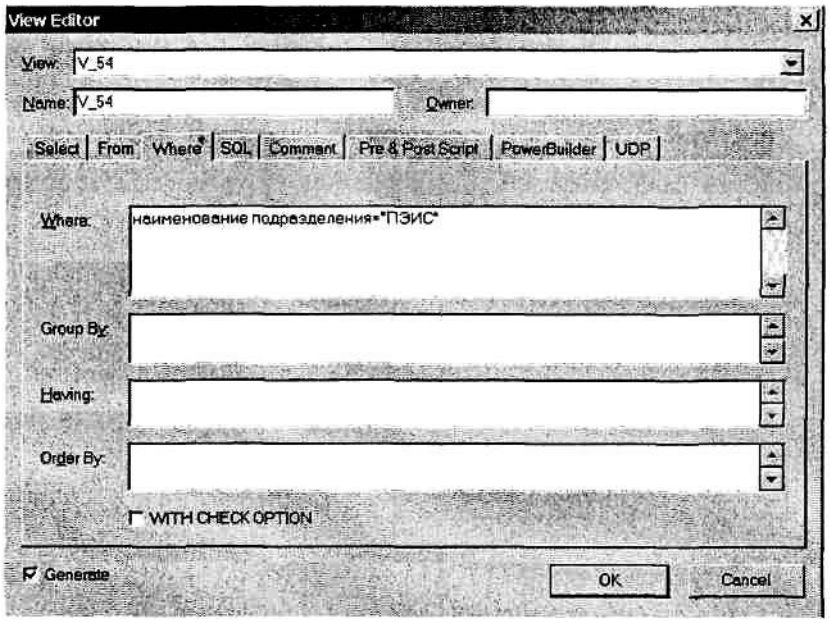


Рис. 7.4. Вид окна редактора представлений.
Задание условий отбора

Возможность создать представление в ERWin имеется для всех целевых СУБД, даже для тех (например, Access), в которых оператор CREATE VIEW в явном виде не поддерживается.

На вкладке SQL (рис. 7.5) можно увидеть текст сгенерированных предложений SQL.

Таким образом, как мы видим, создать представление при использовании ERWin несколько легче, чем при написании оператора CREATE VIEW непосредственно на языке SQL.

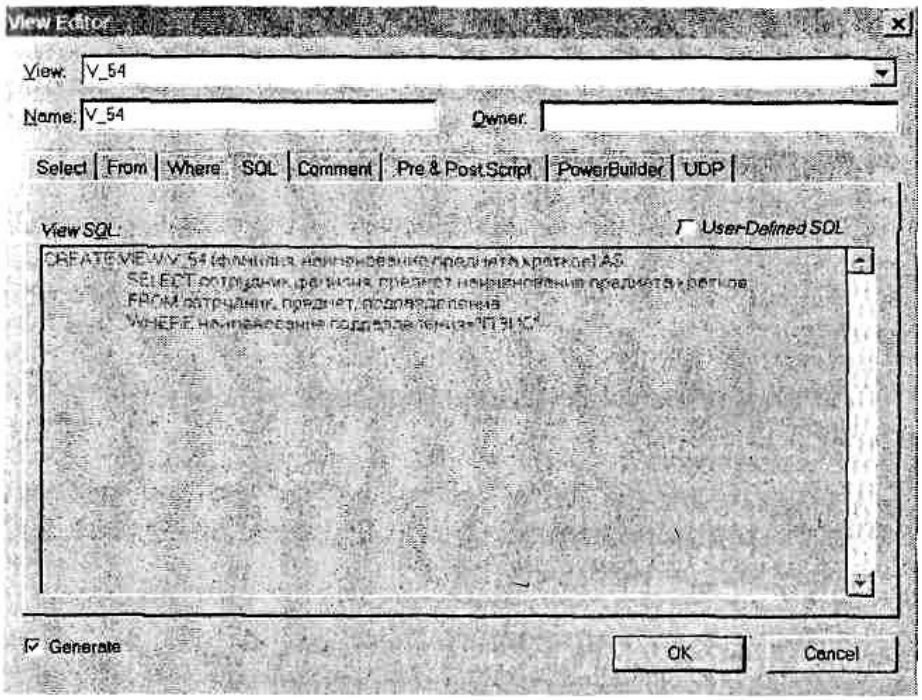


Рис. 7.5. Сгенерированный оператор CREATE VIEW

7.8.4. Курсоры

Курсоры обычно используются в рамках SQL, встроенного в программы, написанные на языке процедурного типа. Использование курсоров вызвано необходимостью стыковать позаписные включающие языки и теоретико-множественный язык SQL.

Курсор определяется посредством оператора объявления курсора (DECLARE CURSOR). Этот оператор определяет имя курсора, а так же содержит в своем составе запрос, выходные данные которого сохраняются для последующей обработки включающей программой:

DECLARE <имя_курсора> **[INSENSITIVE] [SCROLL] CURSOR**
FOR <подзапрос> [<предложение для обновления>]

Курсор находится либо в открытом состоянии, либо в закрытом. Начальное состояние курсора - закрытое. Для открытия курсора используется оператор OPEN CURSOR <имя курсора>. Запрос, описанный при определении курсора, выполняется именно при открытии курсора. При первом открытии курсор устанавливается перед первой строкой. Позиция курсора в открытом состоянии может быть перед определенной строкой, на определенной строке или после последней строки. Если курсор установлен на строку, то эта строка является текущей строкой курсора.

Для построчного извлечения данных из курсора используется оператор FETCH. Этот оператор продвигает позицию открытого курсора на следующую строку в порядке этого курсора, извлекает значения полей этой строки в переменные и имеет следующий синтаксис:

FETCH [[orientation]**FROM]** имя курсора
INTO целевая спецификация.,..;

orientation::=
NEXT | PRIOR | FIRST | LAST | {ABSOLUTE | RELATIVE
спецификация значения}

Значения NEXT, PRIOR, FIRST, LAST понятны без дополнительного пояснения. Значение ABSOLUTE N вызывает перемещение на N-ю строку, а значение RELATIVE N - перемещение на N строк вперед, если N - положительное число, и на N строк назад, если N -отрицательное число.

Если задано предложение INSENSITIVE (нечувствительный), то, пока курсор открыт, все изменения над данными курсора будут игнорироваться.

Строки открытого курсора имеют определенный порядок. Для того чтобы установить нужный порядок, можно использовать в операторе SELECT предложение ORDER BY. В противном случае порядок устанавливается реализацией.

Если задано предложение SCROLL, то не обязательно извлекать строки в том порядке, который задан при открытии.

Курсоры могут быть двух типов: обновляемые и только для чтения. Чтобы курсор был обновляемым, он должен базироваться на одной и только одной таблице, не содержать предложения ORDER BY, не объявляться как INSENSITIVE или SCROLL.

Курсор возвращается в закрытое состояние посредством использования оператора закрытия курсора **CLOSE** <имя курсора.>

7.9. MS Jet Access SQL

Поскольку, несмотря на наличие стандартов, реализация языка в каждой СУБД имеет специфические особенности, а работать приходится в конкретной среде, то примеры на использование SQL необходимо рассматривать в реальных системах. В связи с широким использованием в настоящее время СУБД Access рассмотрим реализацию SQL на примере этой системы.

Microsoft Access SQL в основном отвечает стандарту ANSI-89 (уровень 1). Однако некоторые средства ANSI SQL не применяются в Microsoft Access SQL. И наоборот, Microsoft Access SQL использует зарезервированные слова и средства, не поддерживаемые ANSI SQL.

Кроме того, в Access 2000 имеется режим ANSI SQL-92. Этот режим учитывает расширенный синтаксис SQL и обеспечивает большую совместимость с SQL-92 и Microsoft® SQL Server™. Этот режим доступен только в случае использования программы Microsoft OLE DB Provider для Jet. Новые средства относятся преимущественно к языку определения данных (DDL).

В качестве построителя запросов на SQL можно использовать табличный язык запросов QBE. Но не все типы запросов SQL могут быть реализованы

на QBE. В Access такие типы запросов, как запросы к серверу, управляющие запросы (т.е. запросы, которые корректируют содержимое таблиц) и запросы на объединение, не могут быть определены в бланке запроса. Для создания таких запросов требуется ввести оператор SQL непосредственно в окно запроса в режиме SQL.

7.9.1. Оператор SELECT

Общая характеристика оператора

Оператором SQL, осуществляющим отбор информации из базы данных, является SELECT или его разновидность - SELECT...INTO.

Синтаксис оператора SELECT в Access имеет следующий вид:

SELECT [предикат] {* | таблица.* | [таблица.]поле1
[**AS** псевдоним1] [, [таблица.]поле2 [AS псевдоним2] [,...]]}
FROM выражение [,...] [IN внешняя_база_данных]
[**WHERE**...]
[**GROUP BY**...]
[**HAVING**...]
[**ORDER BY**...]
[WITH OWNERACCESS OPTION]

Как видим, синтаксис оператора SELECT в Access несколько отличается от стандарта. Конструкции [IN внешняя_база_данных] и [WITH OWNERACCESS OPTION] отсутствуют в стандарте SQL.

Ниже перечислены аргументы оператора SELECT.

Аргумент	Описание
Предикат	Один из следующих предикатов отбора: ALL,DISTINCT, DISTINCTROW или TOP. Предикаты используются для ограничения числа возвращаемых записей. Если они отсутствуют, по умолчанию используется предикат ALL
*	Указывает, что выбраны все поля заданной таблицы (таблиц)
Таблица	Имя таблицы, из которой должны быть отобраны записи
Поле1, поле2	Имена полей, из которых должны быть отобраны данные. Порядок следования полей в запросе определяет, в каком порядке соответствующие значения будут выводиться в ответ
Псевдоним1 Псевдоним2	Имена, которые станут заголовками столбцов вместо исходных названий столбцов в таблице
Выражение	Имена одной или нескольких таблиц, которые содержат отбираемые данные

Внешняя_база_данных	Имя базы данных, которая содержит таблицы, указанные с помощью предыдущего аргумента, если они находятся не в текущей базе данных

Имена полей, которые содержат пробелы или знаки препинания, необходимо заключать в квадратные скобки ([ ]).

Описание WITH OWNERACCESS OPTION используется при работе в сети в составе защищенной рабочей группы и здесь рассматриваться не будет.

Предложение SELECT

Предложение SELECT оператора SELECT служит для указания полей, выводимых в результатную таблицу. Это могут быть поля исходных таблиц, вычисляемые поля или поля, содержащие константу. Имена полей, выводимых в ответ, разделяются между собой запятой.

Для отбора всех полей таблицы можно использовать звездочку (*). Следующий оператор отбирает все поля из таблицы «Сотрудник»:

SELECT * FROM Сотрудник;

Если в запросе используется несколько таблиц, то перед знаком (*) следует поместить имя таблицы, из которой будут выводиться все поля, отделенные от этого знака точкой:

SELECT ФИО, Семья.*
FROM Сотрудник **INNER JOIN** Семья
ON Сотрудник.Код_сотрудника = Семья.Код_сотрудника;

В данном примере из таблицы «Семья» в ответ выводятся все поля, а из таблицы «Сотрудник» - только поле ФИО.

Поясним значение предикатов, используемых в качестве аргументов оператора SELECT.

Предикат ALL является значением по умолчанию. Если оператор SQL не содержит ни одного предиката, то считается, что используется предикат ALL. При этом в ответ выдаются все строки результатной таблицы, даже если они совпадают.

Приведенные ниже операторы (1) и (2) SQL эквивалентны; они возвращают все записи из таблицы «Сотрудники»:

1. **SELECT ALL***
   FROM Сотрудники;
2. **SELECT ***
   FROM Сотрудники;

Предикат DISTINCT исключает из результатной таблицы повторяющиеся строки. Предикат DISTINCTROW опускает данные, основанные

на целиком повторяющихся записях. Предикат DISTINCTROW влияет на результат только в случае, если в запрос включены не все поля анализируемых таблиц. Предикат DISTINCTROW игнорируется, если запрос содержит только одну таблицу или все поля всех таблиц.

Поясним разницу между DISTINCTROW и DISTINCT на следующих примерах. Пусть имеется таблица «Расписание»:

Расписание		
Код_сотрудника	Код_предмета	Дата
2	1	01.01.2002
3	1	01.01.2002
2	1	02.01.2002
3	1	02.01.2002
2	2	01.01.2002
3	5	02.01.2002

Использование оператора SELECT с DISTINCTROW

SELECT DISTINCTROW сотрудник.фио, расписание.код_предмета
FROM сотрудник **INNER JOIN** расписание
ON сотрудник.код_сотрудника = расписание.код_сотрудника;

даст следующий результат:

Distinctrow	
ФИО	Кодпредмета
Жуков О. А.	1
Жуков О.А.	2
Жуков О.А.	1
Бурлак Г.Н.	1
Бурлак Г.Н.	5
Бурлак Г.Н.	1

Использование оператора SELECT с DISTINCT

SELECT DISTINCT СОТРУДНИК.ФИО, расписание.код_предмета
FROM СОТРУДНИК **INNER JOIN** расписание
ON СОТРУДНИК.Код_сотрудника = расписание. код_сотрудника;

даст привычный результат, когда в ответ включаются только неповторяющиеся строки результатной таблицы:

Distinct	
ФИО	Код__предмета
Бурлак Г.Н.	1
Бурлак Г.Н.	5
Жуков О.А.	1
Жуков О.А.	2

Если пользоваться QBE как построителем запросов на SQL, то для включения в запрос DISTINCT следует для свойства запроса «уникальные значения» выбрать значение «Да», а для включения в запрос DISTINCTROW необходимо выбрать значение «Да» для свойства запроса «Уникальные записи».

Предикат TOP позволяет возвращать определенное число записей из общего числа записей, получающегося в результате отбора по заданному условию.

Предикат TOP часто используется совместно с предложением ORDER BY для обеспечения возможности вывода в ответ записей, находящихся в начале или в конце некоего диапазона. При этом в зависимости от запроса выбирается нужный вид упорядочения (ASC или DESC).

Следующая инструкция SQL позволяет получить список пяти лучших студентов выпуска 2002 г.:

```
SELECT TOP 5 Имя, Фамилия
FROM Студенты
WHERE ГодВыпуска =2002
ORDER BY СреднийБалл DESC;
```

Предикат TOP не осуществляет выбор между равными значениями. Если в предыдущем примере средние баллы пятого и шестого студента будут равны, то запрос возвратит шесть записей.

Кроме того, можно использовать зарезервированное слово PERCENT для возврата определенного процента записей, находящихся в начале или в конце диапазона, описанного с помощью предложения ORDER BY Предположим, что вместо пяти лучших студентов следует отобрать 10% студентов с наиболее низкой успеваемостью:

```
SELECT TOP 10 PERCENT Фамилия, Имя
FROM Студенты
WHERE ГодВыпуска = 1994
ORDER BY СреднийБалл ASC;
```

Предикат ASC обеспечивает упорядочение по возрастанию. В принципе его можно было не указывать, так как ASC является значением по умолчанию.

Значение, следующее после предиката TOP, должно быть числовым значением без знака типа Integer.

Следует обратить внимание на то, что аргументы DISTINCTROW и TOP, используемые в Access, отсутствуют в стандарте SQL.

Если запрос затрагивает несколько таблиц и в них имеются одноименные поля, перед именами таких полей необходимо ввести имя таблицы и знак « . » (точка).

Предположим, что есть две таблицы: «Продукция» (с полями «Код_продукции», «Наименование_продукции») и «Поставка» (с полями «Код_продукции», «Дата», «Количество»). Поле «Код_продукции» содержится в обеих таблицах. Следующий оператор SQL отберет поле «Наименование_продукции» и «Код_продукции» из таблицы «Продукция» и поля «Дата» и «Количество» из таблицы «Поставка»:

```
SELECT Продукция.Код_продукции, Наименование_продукции,
Дата, Количество
FROM Поставка, Продукция
WHERE Продукция.Код_продукции = Поставка.Код_продукции
AND Наименование_продукции = "кефир";
```

Если не указать имя таблицы при перечислении полей, присутствующих в нескольких таблицах, то будет выдано сообщение об ошибке.

Если создавать запрос SQL, пользуясь QBE как построителем, то имя таблицы будет как префикс указываться перед именем поля всегда, а не только тогда, когда это действительно необходимо. Кроме того, для объединения таблиц всегда используется операция JOIN в предложении FROM. Тот же по смыслу запрос, полученный путем создания запроса на QBE с последующим переходом из режима **Конструктор** в режим **SQL**, будет выглядеть следующим образом:

```
SELECT продукция.Код_продукции,
продукция.Наименование_продукции, Поставка.дата, Поставка.
количество
FROM продукция INNER JOIN Поставка
ON продукция.Код_продукции = Поставка.Код_продукции
WHERE (((продукция.Наименование_продукции)="кефир"))*;
```

При выводе результатов ответа в режиме таблицы имена полей используются в качестве заголовков столбцов. Для изменения заголовка столбца следует использовать зарезервированное слово AS. Использование предложения AS эквивалентно определению свойства «Псевдоним» (Alias) в бланке свойств списка полей в режиме Конструктор запросов.

---

* Скобки сохранены в том виде, в котором они выданы системой при генерации запроса.

Использование псевдонимов особенно важно, когда в БД используются короткие английские названия полей или в ответ выводятся вычисляемые поля. Нижеследующий пример демонстрирует использование псевдонима для вычисляемого поля:

SELECT сотрудник.ФИО, [оклад]*0.5 **AS** Премия
FROM сотрудник;

Предложение FROM

Общая характеристика. В предложении FROM задаются таблицы, участвующие в запросе. Поскольку в Access не поддерживается механизм создания представлений (VIEW)*, то это могут быть либо реальные таблицы БД (причем таблицы и активной БД, и внешней БД), либо ранее запомненные запросы.

Синтаксис предложения FROM:

FROM выражение [**IN** внешняяБазаДанных].

Имена таблиц, участвующих в запросе, могут быть перечислены через запятую. Тогда условие соединения таблиц будет задано в предложении WHERE либо в самом предложении FROM. В последнем случае в качестве выражения используются операции INNER JOIN, LEFT JOIN или RIGHT JOIN.

Предложение FROM должно присутствовать в каждой инструкции SELECT.

Внешняя база данных - это база данных, поддерживающая протокол ODBC, такая, как база данных Microsoft® SQL Server™, обычно расположенная на удаленном сервере. Для повышения быстродействия и облегчения применения рекомендуется использовать присоединение таблиц вместо предложения IN при загрузке данных из внешней базы данных.

Соединение таблиц с использованием операции JOIN. Выше, говоря об определении полей, выводимых в ответ, нам уже пришлось коснуться проблемы соединения таблиц. Рассмотрим этот вопрос более детально. Классическая операция соединения (JOIN) предполагает, что каждая строка первой таблицы по заданному условию сравнивается с каждой строкой второй таблицы, и если условие выполняется, то соответствующие строки сцепляются, образуя очередную строку в результирующем отношении. В принципе условие сравнения может быть любым, но чаще всего используется условие соединения по равенству значений соответствующих полей. Такое соединение называется естественным соединением. В SQL Access данная операция называется INNER JOIN.

* Представление можно создать с помощью нового синтаксиса ANSI SQL CREATE VIEW, доступного через Microsoft® OLE DB Provider для Jet.

Нельзя в качестве полей, по которым проводится соединение таблиц, использовать поля, имеющие тип MEMO или OLE.

Пусть в БД имеются таблицы «Кафедры» и «Сотрудники». В таблице «Кафедры» присутствуют поля «Код_кафедры», «Наименование_ кафедры_полное» и «Наименование_кафедры_краткое». В таблице «Сотрудники» есть поле Код_кафедры, но отсутствуют наименования кафедр. В предметной области возможны ситуации, что некоторые сотрудники не приписаны ни к одной кафедре, и могут быть кафедры, к которым не приписано ни одного сотрудника.

Тогда, если требуется получить список сотрудников кафедр(ы) и в запросе используется наименование кафедры, следует использовать операцию INNER JOIN.

Кроме операции INNER JOIN в SQL Access имеются операции LEFT JOIN и RIGHT JOIN. Операции LEFT JOIN и RIGHT JOIN называются в документации по Access *внешним объединением*[*].

Так, для отбора всех кафедр (в том числе тех, в которых нет ни одного сотрудника) или всех сотрудников (в том числе тех, которые не приписаны ни к какой кафедре) следует использовать операцию LEFT JOIN или RIGHT JOIN соответственно.

Поле, используемое для объединения таблиц, может как включаться, так и не включаться в результат выполнения запроса.

Операция JOIN выглядит следующим образом:

FROM таблица1 **INNER|LEFT|RIGHT JOIN** таблица2 **ON** таблица! поле1 = таблица2.поле2.

Поля связи должны иметь соответствующие друг другу тип данных и длину. Имена полей связи могут различаться, однако при проектировании БД желательно давать одинаковые имена полям, которые имеют одинаковое смысловое значение, и, напротив, поля, различающиеся по смыслу, следует называть разными именами.

Операции JOIN могут быть вложенными. В этом случае используется следующий синтаксис:

SELECT список полей
FROM таблица1 **INNER JOIN**
(таблица2 **INNER JOIN** [(]таблица3
[**INNER JOIN** [(]таблицаX [**INNER JOIN** ...)] ON таблица3.поле3 = таблицаX.полеX)]
ON таблица2.поле2 = таблица3.поле3)
ON таблица1 .поле1 = таблица2.поле2;

[*] Правильнее было бы это называть «соединением», а не «объединением», поскольку в реляционной алгебре операции «соединение» и «объединение» - это две разные операции, и операция JOIN соответствует «соединению».

На рис. 7.6 приведен пример запроса, использующего три таблицы. Для его изображения на SQL будет использована вложенная операция JOIN.

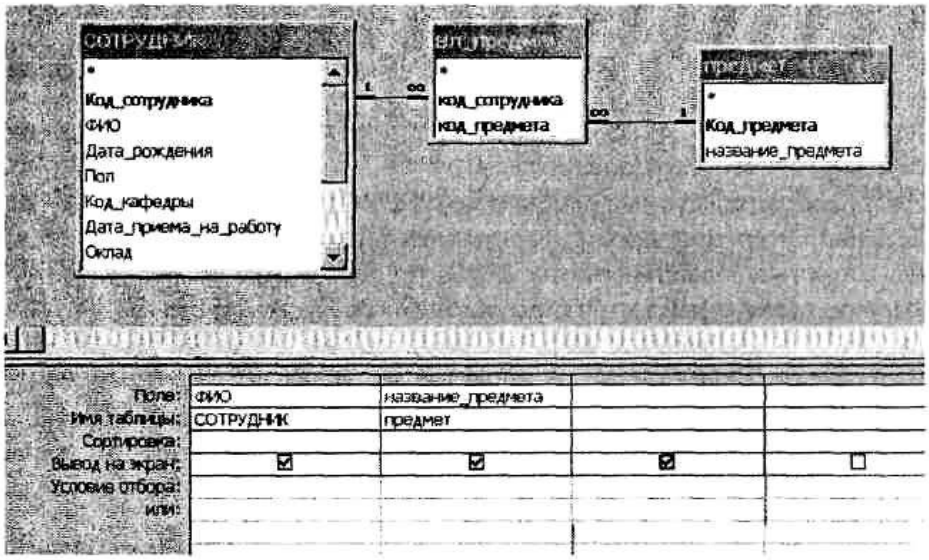


Рис. 7.6. Запрос, использующий три таблицы

Запрос на SQL, соответствующий запросу, изображенному на рис. 7.6, будет выглядеть следующим образом:

SELECT сотрудник.ФИО, предмет.название_предмета_краткое
FROM сотрудник
INNER JOIN (предмет
INNER JOIN вл_предмет ON предмет.Код_предмета = вл_предмет.-код_предмета)
ON сотрудник.Код_сотрудника = вл_предмет.код_сотрудника;

Если не использовать предложение JOIN для выполнения операций объединения нескольких таблиц, результирующий объект Recordset нельзя будет изменять.

Вместо вложенной операции JOIN можно использовать подчиненный запрос.

Предложение WHERE

В предложении WHERE указывается условие отбора. При задании условий отбора используются определенные ограничители, зависящие от типа поля, для которого задается условие отбора. Если при задании запроса на QBE ограничители можно не задавать при вводе запроса (система их введет автоматически), то при задании запроса на SQL необходимо самим правильно определить требуемый ограничитель.

Предложение WHERE может содержать до 40 выражений, связанных логическими операторами.

Имеются некоторые особенности задания запросов в MS Jet SQL. Так, например, при указании аргумента «Условие Отбора» даты должны вводиться в американском формате*, даже если используется неамериканская версия ядра базы данных Jet. Например, дата 23 февраля 1946 г. записывается в России как 23.02.46, а в США как 2/23/46. Даты заключаются в символы «решетки» (#).

В QBE допускается ввод дат в европейском формате. Если запрос на QBE преобразовать в запрос на SQL, то изменение формата даты будет проведено автоматически. Так будет выглядеть запрос, изображенный на рис. 7.7, в формате SQL:

SELECT сотрудник.ФИО, сотрудник.Дата_рождения
FROM сотрудник
WHERE (((сотрудник.Дата_рождения)<#2/23/1946#));

В Microsoft Access SQL конструкции BETWEEN...AND имеет такой же синтаксис, как и в ANSI SQL:

выражение1 [**NOT] BETWEEN** выражение2 AND выражение3

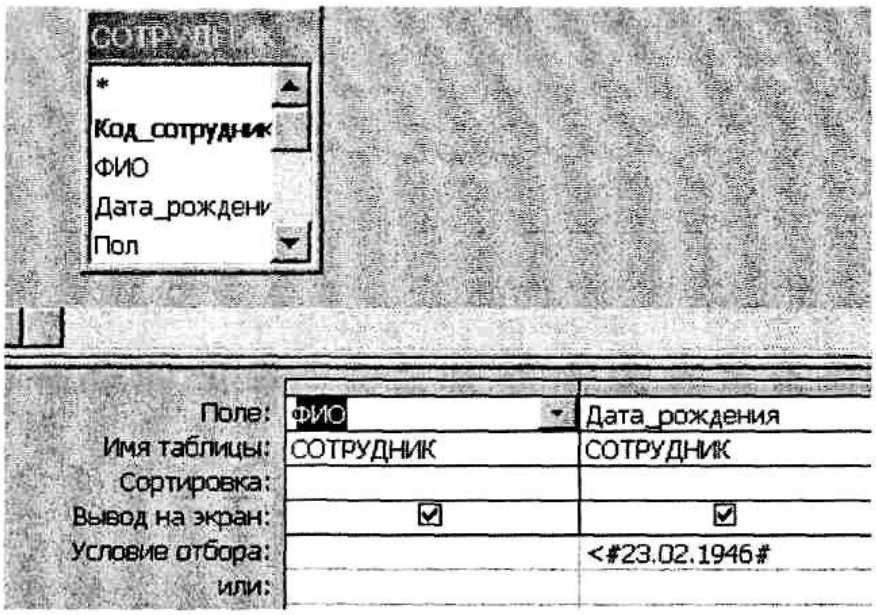


Рис. 7.7. Использование поля типа даты в запросе

Но правила, применимые к ним, различаются: в Microsoft Access SQL «выражение2» может быть больше, чем «выражение3», а в ANSI SQL - нет.

*Американский формат даты означает «месяц-число-год», в то время как европейский формат означает «число-месяц-год».

В приведенном ниже примере выдаются сведения о зарплате сотрудников, чей оклад находится в диапазоне от 1000 до 2000.

SELECT ФИО, оклад
FROM сотрудник
WHERE оклад **Between** 1000 **And** 2000;

Для отбора записей по части значения какого-либо поля используется предикат LIKE. В Microsoft Access SQL в предикате LIKE используются иные символы шаблона, чем в стандарте:

Символ шаблона	Microsoft Access SQL	ANSI SQL
Любой один символ	?	_ (подчеркивание)
Любая группа любых символов	*	%

Ниже приведен запрос, в котором используется предикат LIKE. Он выводит фамилии и автобиографии тех сотрудников, у которых в поле «Автобиография» встречается упоминание о Москве:

SELECT ФИО, автобиография
FROM сотрудник
WHERE автобиография **LIKE** "*Москв*";

Следует обратить внимание на то, подобный поиск может дать некоторый «шум».

При использовании режима ANSI SQL-92 обеспечивается поддержка подстановочных знаков ANSI, т.е. знак процента (%) и символ подчеркивания (_) также являются частью режима ANSI SQL-92. В одном запросе смешивать знаки нельзя.

Предложение GROUP BY

Предложение GROUP BY позволяет группировать записи по одному полю или совокупности полей, указанных в этой фразе. Порядок следования полей в списке имеет значение, поскольку именно он будет определять старшинство признаков группировки. Допускается до десяти уровней группировки. Нельзя осуществлять группировку по полям, имеющим тип MEMO или объект OLE.

Обычно GROUP BY используется вместе со статистическими функциями, позволяющими проводить вычисления для сформированных групп.

Оператор SELECT, содержащий предложение GROUP BY, обычно в качестве полей, выводимых в ответ, включает поля, по которым проводится группировка, и одну (или несколько) статистическую функцию SQL. В

Access используются следующие статистические функции: Avg, Count, First, Last, Min, Max, StDev, StDevP, Sum, Var, VarP. Статистические функции First, Last, StDev, StDevP, Var, VarP в стандарте SQL отсутствуют.

Значения Null, которые находятся в полях, заданных в предложении GROUP BY, группируются и не опускаются. Однако статистические функции SQL значения Null не обрабатывают.

Ниже приведен пример запроса для определения числа сотрудников на каждой кафедре:

SELECT [код кафедры], **Count** (ФИО) **AS** [число_сотрудников]
FROM сотрудник
GROUP BY [код кафедры];

Предложение HAVING

В оператор SELECT, содержащий предложение GROUP BY, могут быть включены как предложение WHERE, так и HAVING. Предложение WHERE используется для исключения записей из группировки, а HAVING - для применения фильтра к записям после группировки.

Предложение HAVING имеет следующий синтаксис:

[**HAVING** условиеГруппировки]

Ниже приведен пример запроса, показывающего число сотрудников на кафедрах с численностью более пяти человек:

SELECT [код кафедры], Count(ФИО) **AS** [число_сотрудников]
FROM сотрудник
GROUP BY [код кафедры]
HAVING Count(ФИО)>5;

Предложение HAVING может содержать до 40 выражений, связанных логическими операторами AND и OR.

Предложение ORDER BY

Предложение ORDER BY используется для сортировки записей, полученных в результате запроса, в порядке возрастания или убывания на основе значений указанного поля (или полей).

Предложение ORDER BY выглядит следующим образом:

[**ORDER BY** попе J [**ASC | DESC**][, поле_2 [**ASC | DESC**]][, ...]]],

где *поле_1, поле_2* - имена полей, по которым сортируются записи, ASC - упорядочение по возрастанию, DESC - упорядочение по убыванию. По умол-

чанию используется порядок сортировки по возрастанию (от А до Я и от 0 до 9).

Для сортировки по убыванию (от Я до А и от 9 до 0) следует добавить зарезервированное слово DESC после имени каждого поля, которое нужно отсортировать в убывающем порядке.

Предложение ORDER BY может содержать несколько полей. Сначала записи сортируются по первому полю в списке ORDER BY.

Затем записи, имеющие совпадающие значения в первом поле, сортируются по второму полю и т.д.

В приведенном ниже операторе SQL записи сортируются по убыванию окладов, а внутри группы одинаковых окладов - по возрастанию ФИО:

SELECT Фамилия, Оклад
FROM Сотрудники
ORDER BY Оклад **DESC**, Фамилия;

Нельзя осуществлять сортировку по полям типа MEMO или объекта OLE.

Предложение ORDER BY обычно является последним элементом инструкции SQL.

7.9.2. Подчиненные запросы SQL

Подчиненный запрос - это оператор SELECT, вложенный в операторы SELECT, SELECT..INTO, INSERT...INTO, DELETE или UPDATE либо в другой подчиненный запрос.

Вложенный запрос в общем случае возвращает какое-то множество записей. Для отбора этих записей во внешнем запросе могут быть использованы:

сравнение [ANY | ALL | SOME] (оператор);
выражение [NOT] IN (оператор);
[NOT] EXISTS (оператор),

где

сравнение - выражение и оператор сравнения, сравнивающий это выражение с результатом выполнения подчиненного запроса;

выражение - выражение, которое должно быть найдено в наборе записей, являющихся результатом выполнения подчиненного запроса;

оператор - правильный оператор SELECT, заключенный в круглые скобки, который и является вложенным запросом.

Подчиненный запрос обычно используется в предложении WHERE или HAVING.

Предикаты ANY или SOME являются синонимами. Они используются для отбора в главном запросе записей, которые удовлетворяют сравнению с какой-либо из записей, отобранных в подчиненном запросе.

В операторе подчиненного запроса SELECT нельзя задавать запрос на объединение или перекрестный запрос.

Ниже в виде вложенного запроса представлен запрос «Список сотрудников, имеющих детей»:

```
SELECT сотрудник.ФИО
FROM сотрудник
WHERE сотрудник.код_сотрудника=ANY (SELECT код_сотрудника
FROM дети);
```

Этот же запрос может быть представлен с использованием предиката IN:

```
SELECT фио
FROM сотрудник
WHERE код_сотрудника IN
(SELECT код_сотрудника
FROM дети);
```

Список сотрудников, не имеющих детей, можно выдать, используя запрос

```
SELECT фио
FROM сотрудник
WHERE код_сотрудника NOT IN
(SELECT код_сотрудника
FROM дети);
```

или запрос с использованием предиката ALL:

```
SELECT сотрудник.фио
FROM сотрудник
WHERE сотрудник.код _сотрудника <>ALL (SELECT код_сотрудника
FROM дети);
```

Тот же запрос с использованием встроенного оператора JOIN может быть представлен в виде

```
SELECT сотрудник.фио
FROM сотрудник LEFT JOIN дети
ON сотрудник.код_сотрудника = дети.код_сотрудника
WHERE ((дети.фио_ребенка) IS NULL);
```

7.9.3. Корректирующие операторы

Добавление

Запрос, приводящий к добавлению новых записей в конец существующей таблицы, называется запросом на добавление. Эти действия можно выполнить при использовании оператора INSERT.

Для добавления одной записи может быть использован оператор

INSERT INTO назначение [(поле_1[, поле_2[, ...]])]
VALUES (значение_1[, значение_2[,...])

Список полей может не задаваться, если значения вводятся во все поля, определенные в структуре таблицы.

Пример 1

INSERT INTO сотрудник
VALUES (32, "Ли", #12/30/189912:14:46#, "м", 1, #12/12/1970#, 2000, 1000, yes, "доцент", 21);

В примере 1 вводятся значения во все поля записи в таблице «Сотрудник», поэтому список полей в предложении INSERT INTO отсутствует.

Пример 2

INSERT INTO сотрудник (фио)
VALUES ("Мун");

Если не все поля определены в операторе INSERT (см. пример 2), в недостающие столбцы будет вставлено значение по умолчанию или значение Null.

В таблицу могут вводиться данные, полученные в результате выполнения запроса. В этом случае оператор выглядит следующим образом:

INSERT INTO назначение [(поле_1[, поле_2[,...]])] [IN внешняя БазаДанных]
SELECT [источник.]поле_1[, поле_2[,...]
FROM выражение

Пример 3

INSERT INTO сотрудник2
SELECT Сотрудник.Код_сотрудника **AS** Код_сотрудника, Сотрудник.ФИО **AS** ФИО, Сотрудник.Дата_рождения **AS** Дата_рождения, Сотрудник.Оклад **AS** Оклад

FROM сотрудник;

Если названия соответствующих полей в целевой таблице и в таблице, получающейся в результате выполнения вложенного SELECT, отличаются друг от друга, то в SELECT можно воспользоваться конструкцией AS, чтобы привести названия полей в соответствие друг другу (в качестве алиасных имен следует задать имена, используемые в целевой таблице). Использование разных имен в целевой таблице и таблице, получаемой в результате выполнения операции SELECT, также возможно. Необходимо только, чтобы они имели соответствующие друг другу типы данных.

Если в предложении SELECT перечислены не все поля, которые имеются в «целевой» таблице, то данные будут добавляться только в те поля, которые обозначены в SELECT.

Запрос на добавление из другой таблицы можно сконструировать, используя табличный язык QBE (см. разд. 6.2).

Обновление

Запрос на обновление изменяет значения некоторых полей указанной таблицы. Записи, которые подвергаются изменению, определяются на основе заданного условия отбора:

UPDATE таблица
SET новоеЗначение
WHERE условиеОтбора

Если «*условиеОтбора*» не задано, то обновляться будут все записи таблицы. При использовании условия отбора, чтобы убедиться, что будут изменены именно требуемые записи, рекомендуется сначала выполнить запрос на выборку, использующий те же самые условия отбора, а затем выполнить запрос на обновление записей.

Запрос

UPDATE сотрудник **SET** оклад =оклад*1,2;

увеличит оклады всех сотрудников на 20%.

Одновременно можно изменить значения нескольких полей записи. В этом случае выражения во фразе SET перечисляются через запятую:

UPDATE Заказы
SET СуммаЗаказа = СуммаЗаказа * 1.1,
СтоимостьДоставки = СтоимостьДоставки * 1.03
WHERE СтранаПолучателя = "Литва";

В рассмотренных выше примерах изменения затрагивали множество записей, а новые значения получались путем вычисления. Изменения могут касаться и одной конкретной записи, а новое значение может быть задано как константа. Например, если сотрудница (Код_ сотрудника=25) вышла замуж и сменила фамилию, то соответствующий запрос может выглядеть следующим образом:

UPDATE сотрудник
SET Фамилия = "Жукова", [семейное положение] = "замужем"
WHERE Код_сотрудника=25;

Имя поля [семейное положение] взято в квадратные скобки в связи с тем, что поле содержит пробел в имени, а для SQL это недопустимо.

В принципе запрос на корректировку может затрагивать несколько связанных таблиц. Например, если мы хотим увеличить зарплату сотрудников, имеющих детей, на 20%, то запрос будет выглядеть следующим образом:

UPDATE DISTINCTROW сотрудник **INNER JOIN** дети
ON сотрудник.Код_сотрудника = дети.Код_сотрудника
SET сотрудник.Оклад = [сотрудник]![Оклад]*1.2;

Запрос

UPDATE сотрудник **INNER JOIN** дети
ON сотрудник.Код_сотрудника = дети.Код_сотрудника
SET сотрудник.Оклад = [сотрудник]![Оклад]*1.2;

приведет к увеличению зарплаты на 20% за каждого из детей.

Записи, измененные с помощью запроса на обновление, часто бывает невозможно восстановить. Поэтому перед выполнением запросов на обновление рекомендуется создавать резервную копию.

Удаление записей

Запрос на удаление записей предназначен для удаления записей из одной (или нескольких взаимосвязанных) таблицы. Для выполнения таких действий используется инструкция DELETE, имеющая синтаксис

DELETE [таблица*]
FROM таблица
WHERE условиеОтбора

Указывать имя таблицы во фразе DELETE необязательно. Если воспользоваться QBE в качестве построителя, то во фразе DELETE перечисляются все поля, которые указываются в условии запроса. Например, запрос на уда-

ление записи, относящейся к аспиранту Сидорову, будет выглядеть следующим образом:

```
DELETE аспирант.ФИО
FROM аспирант
WHERE (((аспирант.ФИО)="Сидоров"));
```

Но лучше не использовать такую форму записи, поскольку она, во-первых, может ввести в заблуждение относительно того, что же удаляется из таблицы, и, во-вторых, загромождает команду, никак не влияя на ее результат. Поэтому лучше эту команду записать следующим образаом:

```
DELETE
FROM аспирант
WHERE ФИО="Сидоров";
```

Запрос на удаление записей из нескольких таблиц выглял внешне как удаление из одной таблицы. Например, запрос

```
DELETE
FROM сотрудник
WHERE сотрудник.Должность="стажер";
```

удалит записи не только из таблицы «Сотрудник», но и из связанны;, с ней таблиц, например из таблицы «Дети», если при задании связи между таблицами задано ограничение связи и определено каскадное удаление записей.

7.9.4. Запрос к серверу

Запрос к серверу SQL - это запрос, который используется для прямой отправки команд на сервер базы данных SQL (например, Microsoft SQL Server). Запросы к серверу позволяют непосредственно работать с таблицами на сервере вместо того, чтобы присоединять их. Запросы к серверу используются для выполнения запросов на языке SQL и системных команд, написанных на варианте языка SQL, который поддерживается на конкретном сервере.

Если запрос к серверу возвращает записи, то они входят в статический набор записей.

Для создания запроса к серверу следует в диалоговом окне **Новый запрос** выбрать режим **Конструктор** и щелкнуть по кнопке **ОК**. Далее в диалоговом окне **Добавление таблицы**, не добавляя таблицы или запросы, щелкнуть по кнопке **Закрыть**. Затем в меню **Запрос** необходимо выбрать команду **Запрос SQL** и подкоманду **Запрос к серверу**, открыть окно свойств запроса и указать в свойстве «Строка подключения ODBC (ODBCConnectStr)» сведения, необходимые для подключения к внешней базе данных. Строку подклю-

чения можно ввести самостоятельно или, щелкнув по кнопке **Построить**, ввести данные о сервере, к которому необходимо подключиться.

В ответ на приглашение сохранить пароль в строке подключения щелкните по кнопке **Да**, если требуется сохранить в строке подключения пароль и имя.

Если запрос не принадлежит к типу запросов, в которых возвращаются записи, выберите для свойства «Возврат записей (Returns-Records)» значение **Нет**.

Введите инструкцию SQL в окно запроса к серверу.

За сведениями о синтаксисе запроса следует обращаться к документации сервера базы данных SQL, на который посылается запрос.

При необходимости Microsoft Access выведет приглашение ввести сведения, требуемые для подключения к базе данных на сервере SQL.

На это следует обратить внимание

- Стандарт SQL - это эталон, к которому следует стремиться при создании СУБД.
- Ни одна СУБД не реализует полностью все возможности стандарта.
- Необходимо детально изучить диалект SQL той СУБД, в среде которой вы работаете.
- Особенно внимательно к особенностям реализации SQL следует относиться при работе в гетерогенной среде.
- SQL не является функционально полным языком. Как правило, он используется совместно с каким-либо включающим языком.

Контрольные вопросы

1. Какие объекты могут быть созданы средствами языка SQL?
2. Какой командой может быть создана таблица? Напишите формат этой команды.
3. Какие типы данных допустимы в SQL?
4. Какие ограничения и каким способом можно задать при описании таблиц в SQL?
5. Можно ли создать таблицы SQL, не пользуясь командой CREATE? Если да, то как это можно сделать?
6. Какой командой можно удалить таблицу? Напишите формат этой команды.
7. Что происходит при удалении таблицы с объектами, связанными с этой таблицей?
8. Какой оператор используется для ввода новых данных в таблицу? Напишите формат этого оператора.
9. Как будет выглядеть оператор команды INSERT при вводе в таблицу значений не всех колонок?
10. Как будет выглядеть формат оператора INSERT при вводе в таблицу зна-

чений всех колонок в том порядке, в котором они заданы при описании таблицы?
11. Как можно добавить в таблицу данные из другой таблицы? Какие условия должны быть выполнены при добавлении записей из другой таблицы?
12. Какой оператор используется для корректировки значений отдельных колонок таблицы? Напишите формат этого оператора.
13. Можно ли задать корректировку всех значений определенной колонки таблицы? Если да, то как это можно сделать?
14. Как можно удалить заданную строку в таблице? Напишите формат соответствующего оператора.
15. Что произойдет, если в операторе DELETE не будет задана фраза WHERE?
16. Напишите полный формат оператора SELECT.
17. Какие предложения являются обязательными в операторе SELECT?
18. Как можно задать состав колонок, выводимых в ответ?
19. Как можно вывести в ответ значения вычисляемых колонок?
20. Какие имена будут иметь вычисляемые колонки? Как можно изменить имя выводимой колонки?
21. Как можно коротко указать, что в ответ должны быть выведены значения всех колонок таблицы?
22. Что означает параметр ALL в предложении SELECT?
23. Что означает параметр DISTINCT в предложении SELECT?
24. Какие агрегирующие функции могут быть использованы в SQL?
25. Могут ли в предложении SELECT использоваться агрегирующие функции? константы?
26. В каких предложениях оператора SELECT могут использоваться агрегирующие функции?
27. В каком предложении оператора SELECT указывается, из каких таблиц должны извлекаться данные?
28. Назовите все известные вам способы задания совместной обработки нескольких таблиц.
29. Каково назначение фразы WHERE в операторе SELECT?
30. Назовите все известные вам способы, которыми можно задать выбор значений по заданному диапазону.
31. С какой целью используется предикат LIKE? для каких колонок?
32. Что называется вложенным запросом? Как он реализуется?
33. В каких случаях надо использовать предложение GROUP BY в команде SELECT?
34. Как связаны между собой предложения SELECT и GROUP BY команды SELECT?
35. Как связано между собой использование предложения GROUP BY и агрегирующих функций?
36. Каково назначение фразы HAVING?
37. Могут ли в команде SELECT одновременно использоваться предложения WHERE и HAVING?

38. Как можно задать упорядочение данных в ответе?
39. Как связаны между собой предложения SELECT и ORDER BY в операторе SELECT?
40. Как можно выполнить соединение нескольких таблиц в SQL?
41. Как можно выполнить объединение нескольких таблиц в SQL?
42. Какие ограничения накладываются на операторы SELECT, результаты которых объединяются?
43. Как можно запомнить результаты выполнения запросов?
44. Какие виды таблиц различают в SQL?
45. Что такое «представление»?
46. Как можно создать представление? Напишите формат соответствующего оператора.
47. С какой целью используются представления?
48. Какие представления являются обновляемыми?
49. Что представляют собой курсоры? Как они создаются? В каких случаях и для каких целей используются?
50. Какие отличия имеет MS Jet Access SQL от стандарта ANSI?

Глава 8 СОЗДАНИЕ ЭКРАННЫХ ФОРМ И СТРАНИЦ ДОСТУПА

8.1. Понятие, классификация и роль экранных форм

Ввод и просмотр данных в режиме таблицы имеет много очевидных недостатков. Поэтому, как правило, для удобства пользователей создаются экранные формы, которые позволяют рационально расположить данные на экране, использовать разнообразные элементы оформления, обеспечивать возможности специфических проверок при вводе данных. Во многих СУБД имеются возможности использования Мастеров, автоматизирующих процесс создания экранных форм. Генераторы экранных форм являются компонентом языков 4-го поколения. Использование генераторов экранных форм позволяет практически без программирования создавать довольно сложные системы обработки данных с хорошим пользовательским интерфейсом.

Экранные формы уже давно применяются в практике создания БнД. Роль форм очень сильно возросла и изменилась в последнее время. Сейчас экранные формы используются не только для ввода данных в базу данных, корректировки данных, а также позаписного вывода данных, что является их традиционными сферами применения, но и для реализации просмотров БД по определенным условиям, и даже создания «заставок» и меню (так называемые несвязанные формы, т.е. формы, не связанные с какой-либо таблицей).*

Если первоначально формы определялись только как способ отображения одной записи из БД, то сейчас в формах допускается наличие табличной (многострочной) части; более того, допускается наличие множества «динамических» частей, возможно, из разных БД.

Тем не менее в некоторых СУБД «форма» так и сохранила свое назначение для позаписной обработки БД.

Понятие «форма» и инструменты ее создания используются сейчас не только в СУБД, но и в других классах программных систем, например в табличных процессорах.

Общепринятой в настоящее время является практика, когда в экранной форме помимо информационных присутствуют и управляющие (controls) элементы.†

Несмотря на наличие возможности создавать несвязанные формы, чаще всего формы все-таки связаны с таблицами БД (одной или несколькими взаимосвязанными). Кроме того, экранные формы могут создаваться на ос-

---

* Так, например, в Access включено приложение «Решения», в котором иллюстрируются различные приемы работы с системой, и именно несвязанная форма исполняет роль Главного меню и служит для обращения к тем или иным формам и документам.

† Здесь под управляющими элементами будем понимать только те компоненты формы, которые выполняют какое-то действие над отдельными элементами формы или над всей формой, или над записями БД при манипулировании ими (например, кнопки, обеспечивающие переход к записи, выбор вводимого значения, закрытие формы и т.п.).

нове предварительно сформулированных запросов; для таких форм их связь с таблицами является опосредованной.

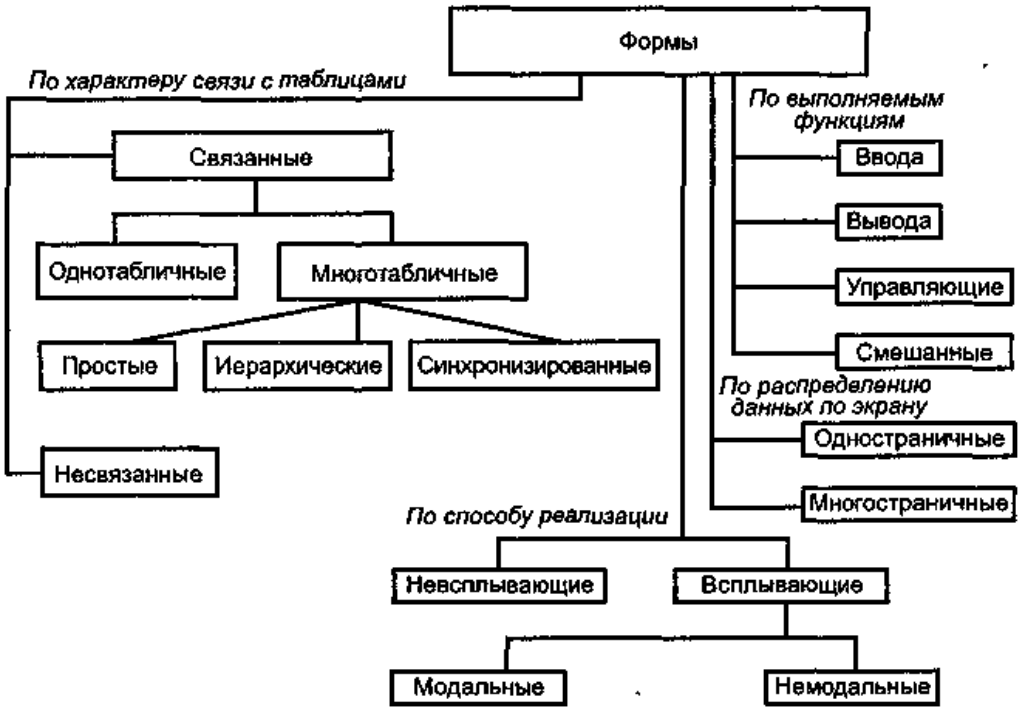


Рис. 8.1. Классификация экранных форм

Экранные формы можно классифицировать (рис. 8.1) по ряду признаков.

1. По **характеру связи с таблицами** различают связанные и не связанные экранные формы. Если форма отражает какие-либо данные из таблиц баз данных, она называется связанной (или присоединенной), в противном случае - *несвязанной*.

По *числу используемых таблиц* выделяют *однотабличные* и *многотабличные* формы.

По характеру соподчинения отдельных частей многотабличные формы классифицируются как *простые*, *иерархические* и *синхронизированные*. Простые многотабличные формы хотя и содержат данные из разных таблиц, но не имеют в своем составе соподчиненных частей. Такие формы могут возникнуть, например, когда они базируются на таблицах, связанных друг с другом отношением 1:1, или когда в форму выводятся данные из таблиц, связанных друг с другом отношением 1 :М, но в форму в основном выводятся данные, находящиеся со стороны «М», а из таблицы, находящейся со стороны «1», берутся какие-то, обычно справочные, данные, т.е. ведущим здесь как бы является таблица, находящаяся со стороны «М». Но наиболее естественной для многотабличных форм все-таки является ситуация, когда ведущая таблица находится на стороне «1». В этом случае создаются иерархические формы, когда в форму в ее общей части выводятся данные из одной записи ведущей таблицы, а в табличной части - множество связанных с ней записей ведомого (зависимого) файла. Иногда (по разным причинам) бывает

нецелесообразно выводить в одну иерархическую форму данные и из основного, и из зависимого файла, и данные из зависимого файла выводятся в отдельной «зоне», которая открывается «при нажатии» соответствующей управляющей кнопки. Такие формы называются синхронизированными.

2. По **выполняемым функциям** различают формы *ввода, вывода, управляющие, смешанные*. Назначение каждого вида этих форм ясно из их названия. Следует обратить внимание на то, что даже в случае, если формы для ввода и вывода полностью совпадают по своему внешнему виду, иногда целесообразно их выполнить как самостоятельные формы в целях обеспечения безопасности данных.

3. По **распределению данных по экранам** (страницам) формы делятся на *одностраничные* и *многостраничные*; одной из разновидностей многостраничных форм можно считать *формы с вкладками*.

4. По **способу реализации** экранные формы могут быть *всплывающими* и *невсплывающими*. Всплывающая форма располагается поверх других открытых форм, даже если активной является другая форма. Всплывающая форма может быть немодальной или модальной. Если всплывающая форма - модальная, пользователь имеет возможность получить доступ к другим объектам и командам меню, пока форма открыта. Если всплывающая форма является немодальной, нельзя получить доступ к любым другим объектам или командам меню, пока форма открыта. Пользователь должен выполнить какое-либо действие, чтобы фокус был переключен на другую форму (или окно).

5. По **форме представления информации** экранные формы могут содержать символьную информацию, деловую графику, информацию, представленную в мультимедийной форме.

Например, в БД, хранящей информацию о животных, наряду с описанием каждого вида может выводиться изображение животного и воспроизводиться издаваемый им звук.

Несмотря на такое широкое применение экранных форм для реализации разных целей, основное внимание далее уделим их использованию для организации ввода данных в БД, потому что, во-первых, это является одним из основных назначений экранных форм; во-вторых, именно этот аспект наиболее значим для процессов создания и ведения БД; в-третьих, такие функции, как создание меню, вывод информации из БД и т.п., могут выполняться с использованием и других средств СУБД.

8.2. Рекомендации по созданию форм

Как было отмечено выше, существуют разные типы экранных форм, и некоторые требования/рекомендации по созданию форм относятся ко всем типам форм, а некоторые - только к отдельным их разновидностям.

Рассмотрим основные рекомендации по созданию форм.

1. Порядок размещения элементов в форме будет зависеть от типа и назначения формы.

Порядок расположения полей на экране для форм, использующихся для вывода информации, определяется в основном смысловой группировкой информации, удобством для восприятия. Так как информационные потребности разных пользователей могут различаться, то в принципе на основе одного и того же источника (таблицы, запроса) может быть создано несколько разных экранных форм, ориентированных на разные запросы пользователей, с соответствующим составом и порядком следования полей.

Для форм, предназначенных для ввода данных в БД, порядок размещения полей ввода на экране также может не совпадать ни с порядком полей в структуре базы данных, ни с порядком их ввода (определяемого обычно расположением соответствующих данных во входном документе, с которого проводится ввод данных). Порядок же перемещения курсора при вводе данных должен соответствовать порядку следования вводимых полей в исходном документе. Если порядок размещения полей на экране не соответствует требуемому порядку ввода данных, то некоторые СУБД дают возможность изменять (задавать) порядок ввода данных (т.е. последовательность позиционирования курсора при переходе от одного элемента к другому). Если СУБД не позволяет задать произвольный порядок перемещения курсора между полями, то порядок следования полей в форме ввода должен соответствовать последовательности их ввода. Вообще, даже если возможность задавать порядок обхода полей есть, лучше, чтобы для форм, использующихся для ввода данных, порядок расположения полей в форме совпадал с порядком их расположения во входном документе.

2. Если форма предназначена для ввода данных, то из нее могут быть исключены поля, которые автоматически вводятся в БД и не могут быть изменены пользователем (например, поле счетчика, вычисляемые поля).

3. Средства современных СУБД обладают разнообразными возможностями по оформлению экрана. При выборе стиля оформления экрана желательно выполнять следующие рекомендации:

а. не стоит злоупотреблять использованием цветов, шрифтов и других оформительских эффектов; оформление экрана не должно отвлекать от выполнения основных функций;

б. яркие цвета (например, красный) лучше использовать только для целей привлечения внимания (например, при сигнале о существенной ошибке);

в. следует быть осторожными при изменении размера шрифта для значений полей, так как при его увеличении текст может не уместиться в отведенное ему окошко и пользователю придется вручную подгонять поля к новому размеру текста;

г. на разных экранах одного приложения рекомендуется использовать единый стиль, выражающийся не только в оформлении, но и в расположении информации; обычно верхняя часть экрана используется как заголовочная, средняя - как основная рабочая зона, нижняя - для размещения итоговых данных, если они имеются, а также в качестве зоны вывода подсказок;

д. не следует перегружать форму большим числом элементов; при необ-

ходимости размещения в форме большого количества данных лучше распределить информацию между несколькими «страницами», чем нагромождать множество элементов на одном экране, использовать мелкий шрифт и т. п.

4. В настоящее время значительно возросли возможности задания настройки формы во время ее использования. Так, некоторые системы (например, Access) позволяют для отдельных элементов формы указывать, выводится этот элемент всегда, только на экран или только на печать (свойство «режим вывода»). Возможно создание форм, которые могут во время работы переключаться для выполнения нужной функции (например, ввода или редактирования данных). И если раньше, отчасти из-за сложности реализации, при проектировании ИС обычно создавались отдельные формы для каждой операции, использующей их для работы с БД (для ввода данных, корректировки отдельных полей, для просмотра и вывода информации на экран или печать), то сейчас появились возможности сравнительно просто реализовывать сложные многофункциональные формы. Но при создании таких форм не следует забывать данные выше рекомендации (см. п. 3д).

5. В формах, используемых для ввода информации, обычно задаются ограничения целостности. Во многих современных СУБД имеется возможность задавать ограничения целостности различными способами. При проектировании системы необходимо тщательно спланировать, какие ограничения целостности и каким способом будут задаваться.

Возможности задания ограничений целостности при описании таблицы могут при «несистемном» проектировании привести к нежелательным последствиям. Предположим, вы создали несколько форм для ввода данных в одну таблицу (например, первая форма -для ввода данных с одного документа, вторая форма - с другого), а в таблице определено несколько обязательных полей, часть из которых должна вводиться посредством первой формы, а другая часть - посредством второй формы. В этом случае частичный ввод данных из одной формы, а потом добавление данных из другой будет невозможно выполнить.

В силу имеющихся ограничений целостности часть полей является обязательной для ввода, другая - нет. Для полей, обязательных для заполнения, можно использовать специальное цветовое выделение.

6. Применение экранных форм для ввода данных позволяет не только соответствующим образом оформить экран, обеспечить контроль правильности данных, но также использовать приемы для рационализации процесса ввода данных: выбор из списка, автоматический перенос данных из других таблиц (чаще всего это ключевые поля из связанной основной таблицы), задание значений по умолчанию, использование шаблонов и др. Причем выбор типа экранной формы будет влиять на возможности организации ввода информации.

7. Во всех формах одного приложения должно быть обеспечено единообразное выполнение одинаковых действий. Например, переход к следующим/предыдущим полю, записи, странице формы должен осуществляться с использованием одних и тех же клавиш при выполнении аналогичных дейст-

вий в разных формах или одинаково оформленных и расположенных кнопок в экранных формах.

8.3. Создание экранных форм в СУБД Access

8.3.1. Выбор способа создания формы

Как было отмечено в главе 5, после описания таблицы можно сразу вводить в нее данные. Но такой способ имеет много очевидных недостатков, поэтому для целей ввода данных обычно используются так называемые экранные формы.

Форму можно создавать несколькими способами. Для того чтобы создать новую форму, необходимо выбрать вкладку **Формы** в окне базы данных и щелкнуть по кнопке **Создать** - появится окно **Новая форма** (рис. 8.2).

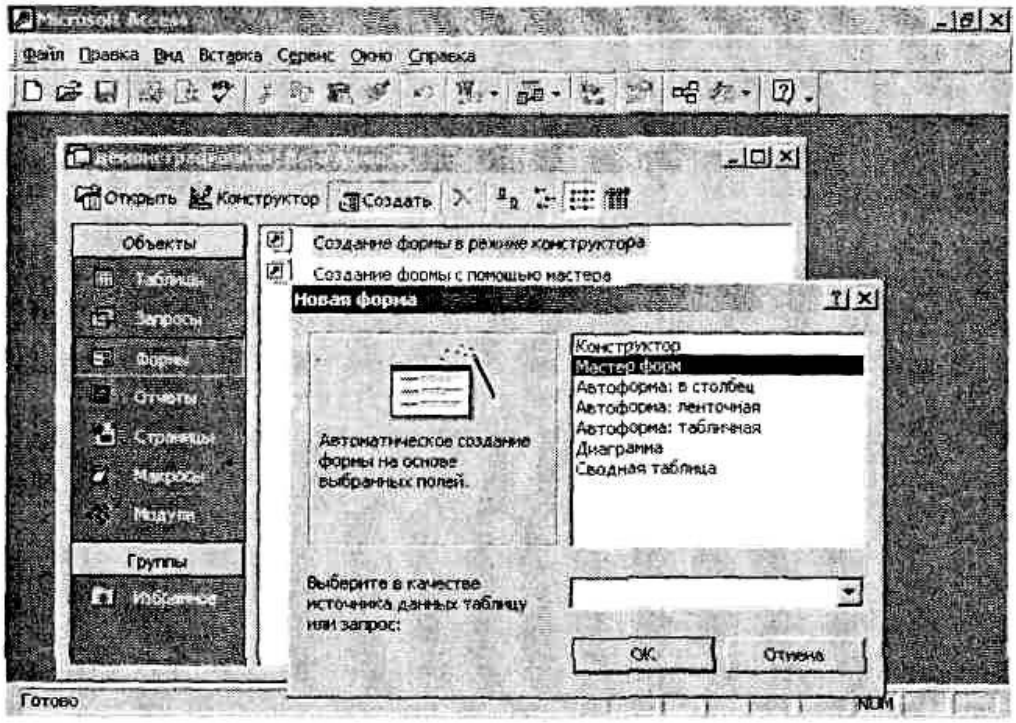


Рис. 8.2. Начальный экран создания форм

8.3.2. Создание форм с помощью Мастера

Создание простой связанной формы с помощью Мастера

При создании формы, связанной с таблицами базы данных, лучше сначала воспользоваться помощью Мастера, а потом модифицировать полученную таким образом форму. Это ускорит и упростит процесс создания форм.

В окне **Новая форма**, кроме выбора способа создания/вида формы можно определить также источник данных для создаваемой формы (см. рис. 8.2). Из этого окна можно выбрать только одну таблицу в качестве источника данных для формы. В качестве примера создадим форму для таблицы «Со-

трудник». Выбор источника можно осуществить и на следующем шаге (рис. 8.3) в окне **Таблицы и запросы**.

Если источником должны являться несколько таблиц, то можно использовать разные способы, которые будут рассмотрены позднее.

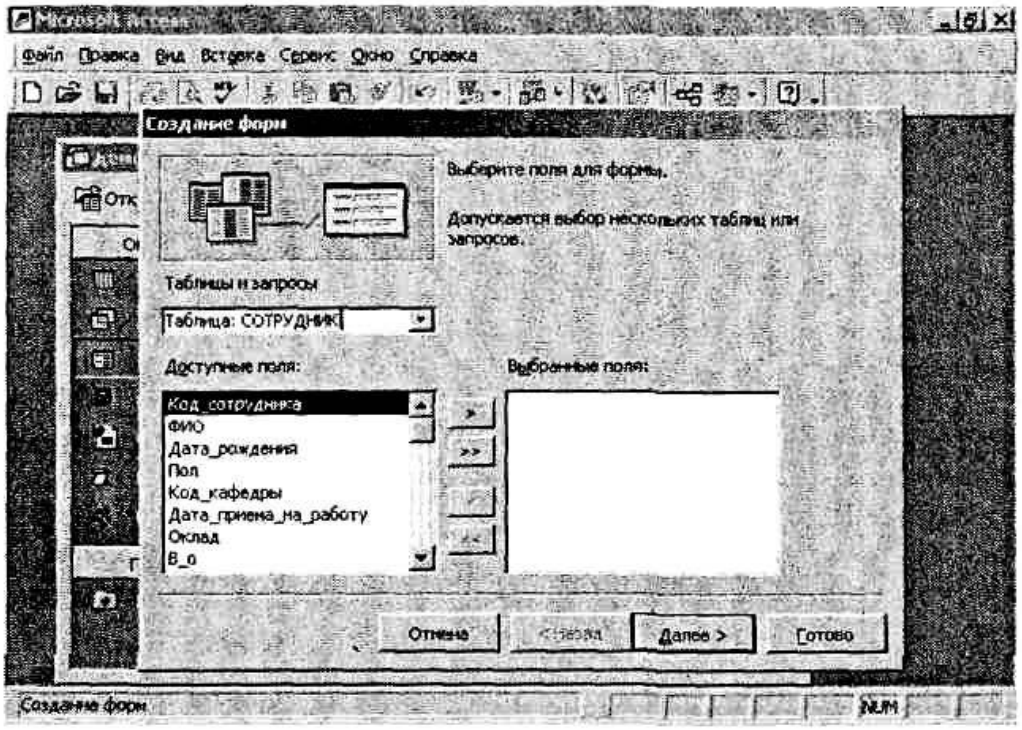


Рис. 8.3. Выбор полей, включаемых в форму

Следующим шагом при создании форм является выбор полей, которые будут включены в экранную форму (см. рис. 8.3). Поля могут переноситься в форму по одному и все сразу. В первом случае необходимо позиционироваться на нужное поле и щёлкнуть по кнопке с одинарной стрелкой, направленной вправо. Чтобы перенести все поля, следует воспользоваться кнопкой с двойной стрелкой. Поля, включенные в форму, на этапе определения состава полей можно исключить, воспользовавшись кнопками со стрелками, направленными влево.

При использовании Мастера для создания формы выбор таблицы и хотя бы одного поля обязателен.

Источником данных для формы могут быть не только таблица (таблицы), но и запросы. Напомним, что в запросах можно задействовать знак «звездочка» (*). При его использовании все поля исходной таблицы выводятся в ответ. Если уже после создания такого запроса в таблицу будут добавлены новые поля, то они при выполнении запроса будут попадать в ответ. Если же на основе такого запроса создана экранная форма, то в нее войдут те поля, которые на момент создания экранной формы содержались в таблице. Поля, которые вставлены в таблицу после создания экранной формы, в экранной форме отражаться не будут. Поэтому, если необходимо, чтобы все поля ис-

ходной таблицы были отображены в форме, созданную форму придется корректировать вручную.

Предположим, что мы хотим создать форму на основе таблицы «Сотрудник» и включить в эту форму все поля исходной таблицы.

После определения состава полей, включенных в форму, следует выбрать внешний вид формы (рис. 8.4). Удобнее сначала выбрать вид «в один столбец»*, а затем разместить элементы формы по экрану так, чтобы это было удобно для последующей работы с формой.

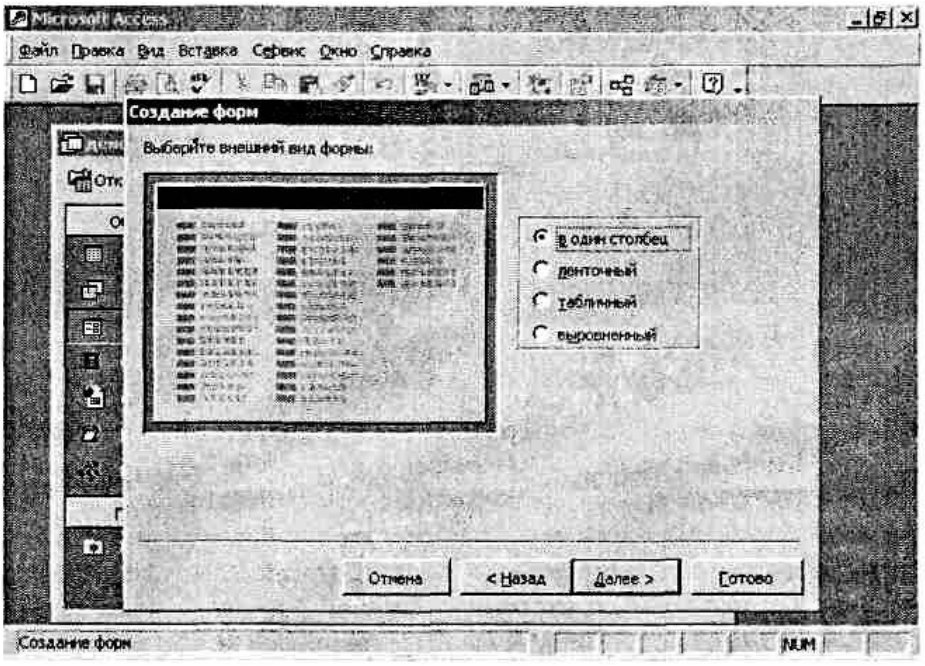


Рис. 8.4. Выбор внешнего вида формы

Следующим шагом является выбор стиля формы (рис. 8.5). Стиль экранной формы позволяет выбрать цвет и «фактуру» формы, т.е. решить чисто оформительские задачи.

Создание формы завершается заданием ее имени (рис. 8.6). По умолчанию дается имя, совпадающее с именем источника данных.

* Название вида формы «в один столбец» несколько условно. Действительно, при данном выборе создается экран «анкетной» формы: одна под другой располагаются пары «название поля/ содержание поля». Но когда полей много и они все не умещаются в один столбец на экране, система автоматически помещает их в два столбца, три и т.п.

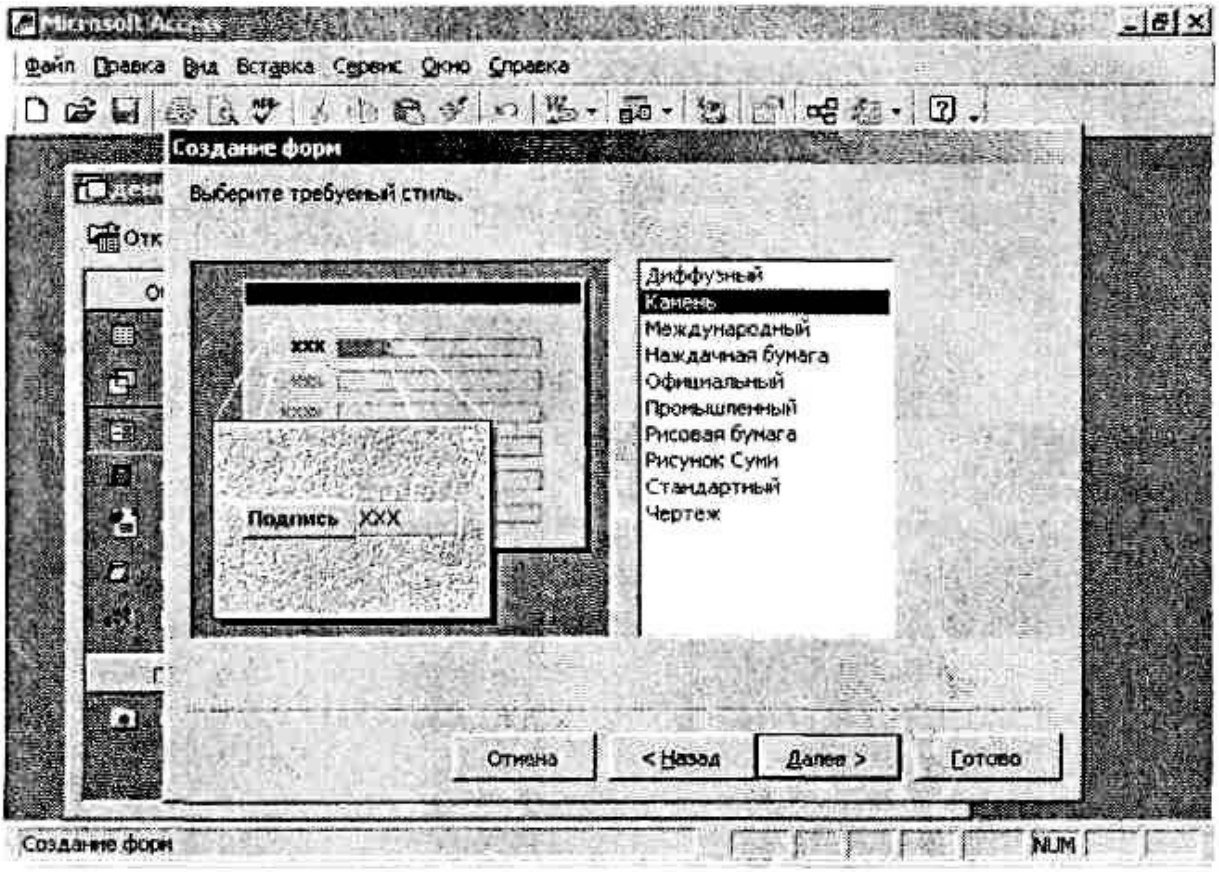


Рис. 8.5. Выбор стиля формы

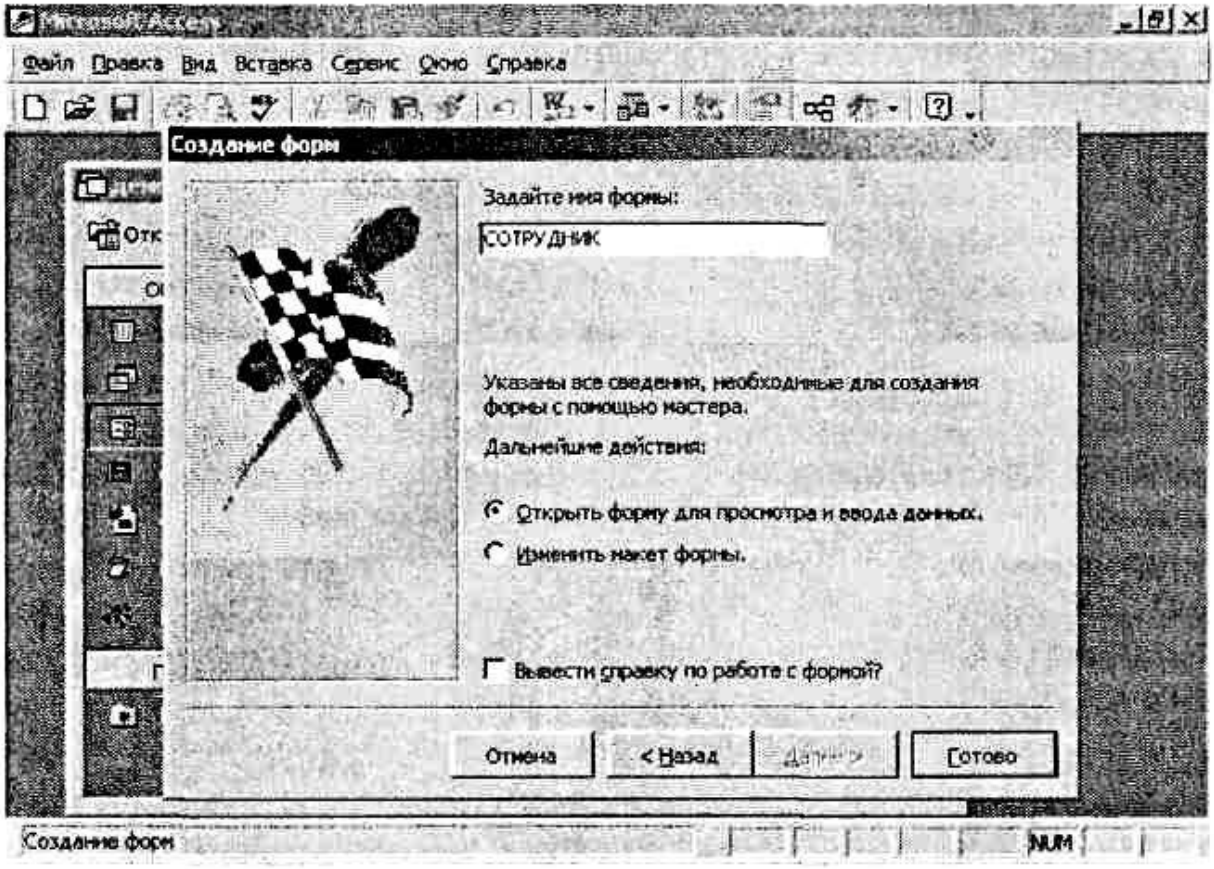


Рис. 8.6. Задание названия формы

Созданная экранная форма может не полностью соответствовать потребностям разработчика. Чтобы ее подправить, можно сразу перейти в режим **Конструктор**, выбрав альтернативу «Изменить макет формы», а можно это сделать и потом, перейдя в режим **Конструктор** из режима формы или открыв ранее созданную форму в режиме **Конструктор**.

Создание многотабличной формы с помощью Мастера

Многотабличную форму также можно создать с помощью Мастера. При создании форм, источником данных для которых являются несколько таблиц, можно, находясь в окне **Создание форм** (см. рис. 8.3), последовательно выбирать таблицы, являющиеся источником данных, и поля из них. При этом таблицы должны быть обязательно предварительно связаны между собой.

Внимание! Порядок, в котором выбираются таблицы при создании «многотабличной формы», имеет большое значение.

Предположим, что мы хотим создать экранную форму, содержащую сведения из таблиц «Сотрудник» и «Дети». Если выберем сначала таблицу «Сотрудник» и из нее - поле «ФИО», а затем таблицу «Дети» и из нее - все поля, то создастся составная форма, которую удобно использовать для ввода данных в таблицу «Дети». Создание многотабличной формы начинается так же, как и создание однотабличной формы. На первом шаге выбираем способ создания формы -«Мастер форм» и таблицу-источник - «Сотрудник». На втором шаге из таблицы «Сотрудник» выберем поле «ФИО». После этого вернемся к окошку «Таблицы и запросы» и выберем в нем вторую, соподчиненную таблицу - «Дети» (рис. 8.7). Перенесем все поля из этой таблицы в форму.

Дальнейшая последовательность шагов создания составной формы представлена на рис. 8.8 - 8.10. ,

Сначала выбирается вид представления данных - подчиненная или связанная форма (рис. 8.8). Для наших целей подходит подчиненная форма.

Затем выбирается вид подчиненной формы (рис. 8.9).

Следующий шаг (рис. 8.10 - выбор стиля) совпадает с аналогичным шагом при создании однотабличной формы.

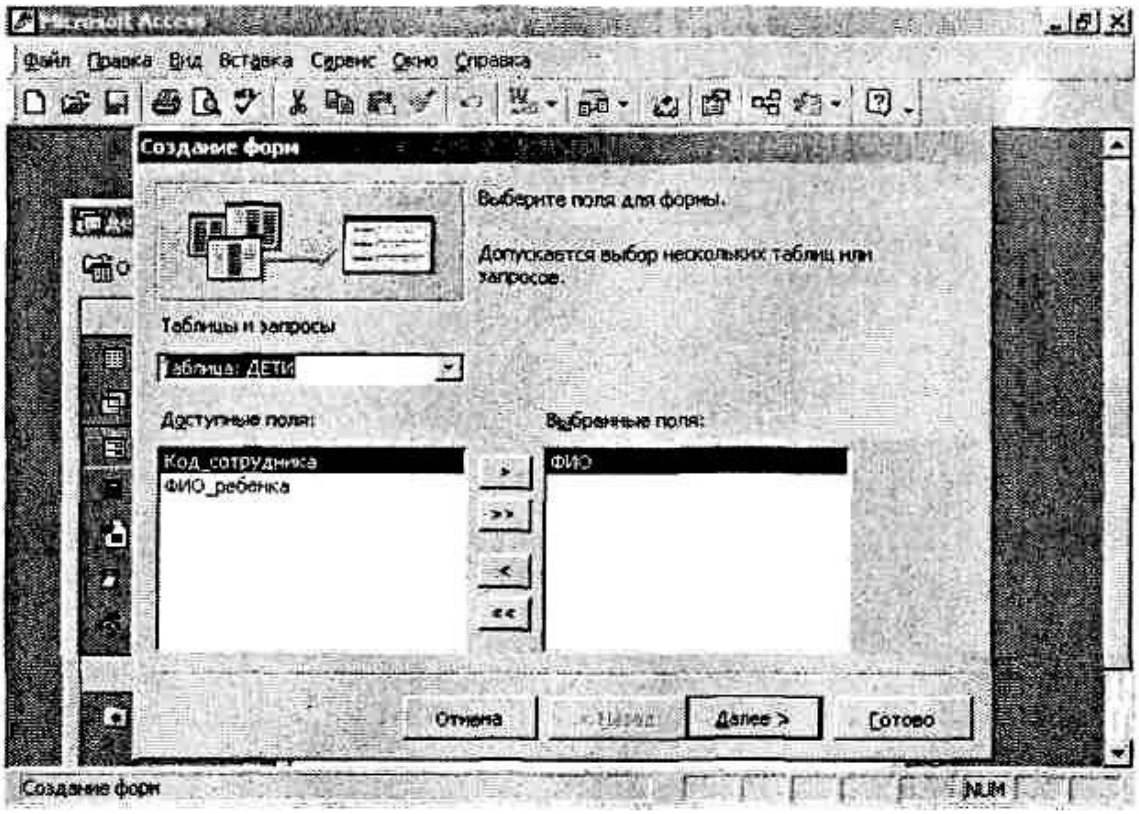


Рис. 8.7. Создание многотабличной формы (выбор полей)

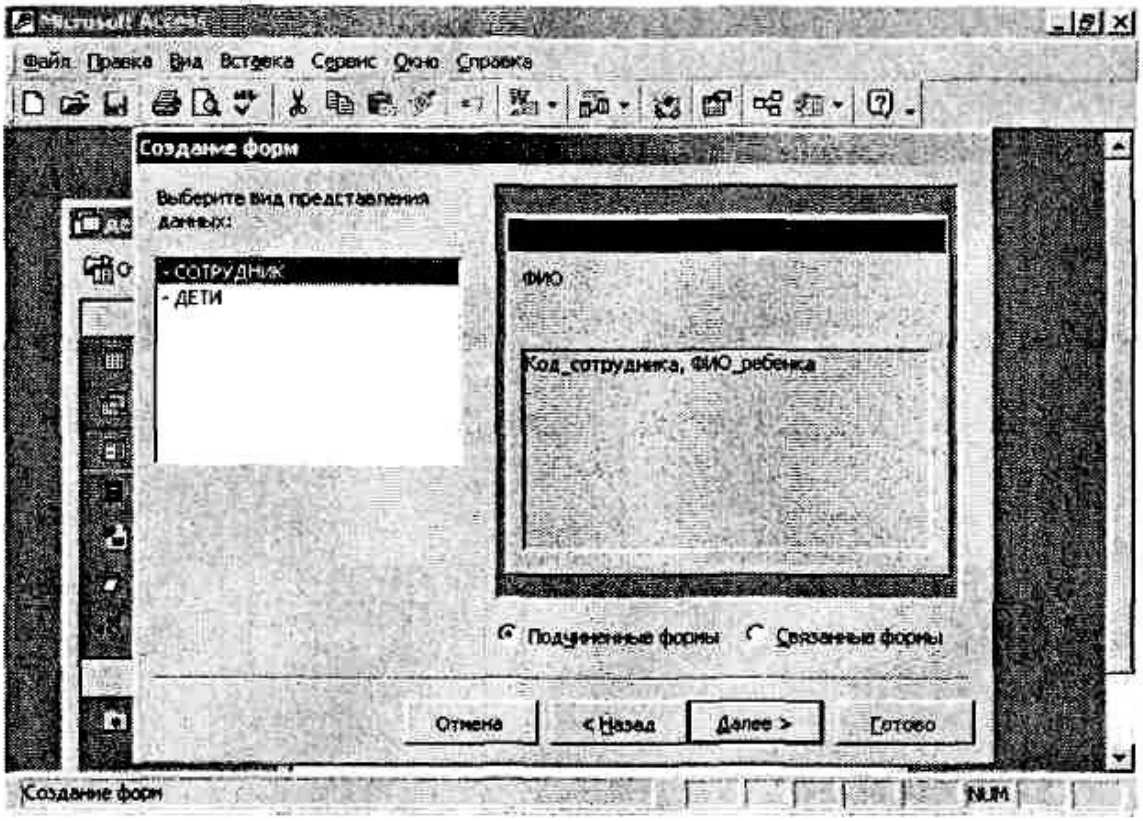


Рис. 8.8. Создание многотабличной формы (выбор вида представления)

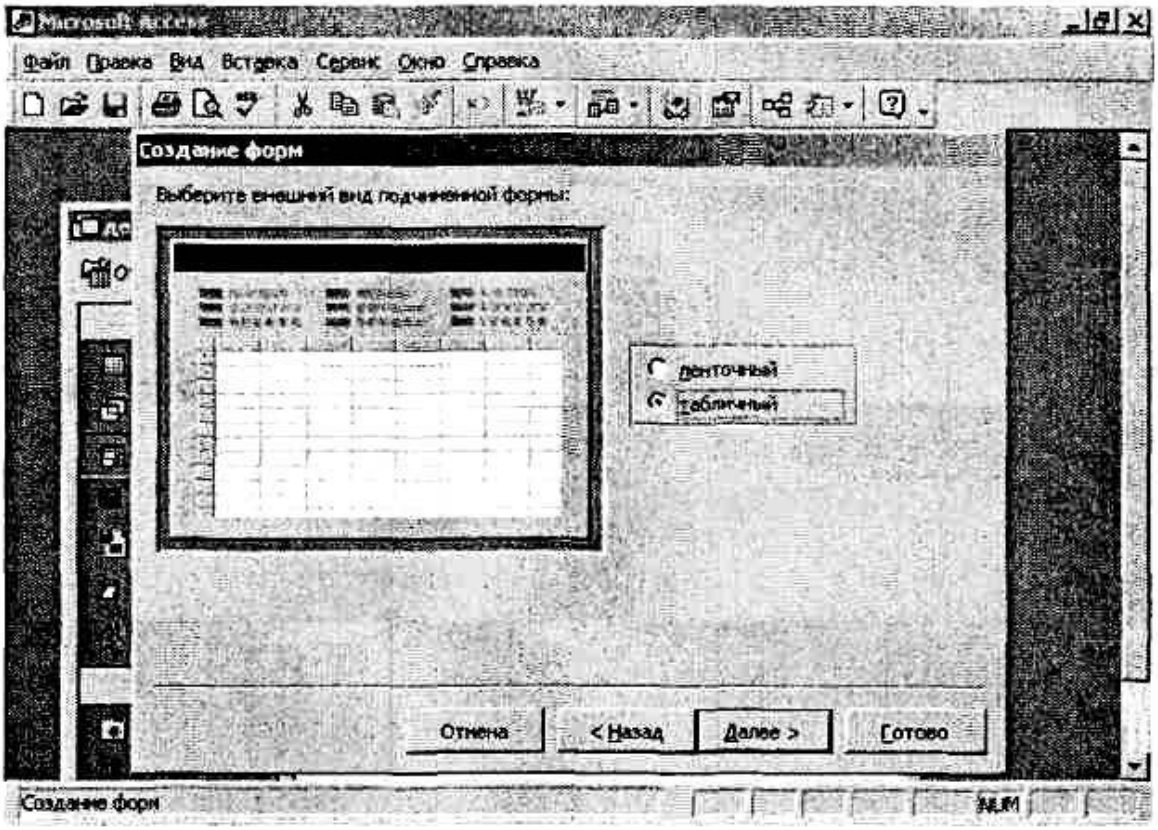


Рис. 8.9. Создание многотабличной формы (выбор вида подчиненной формы))

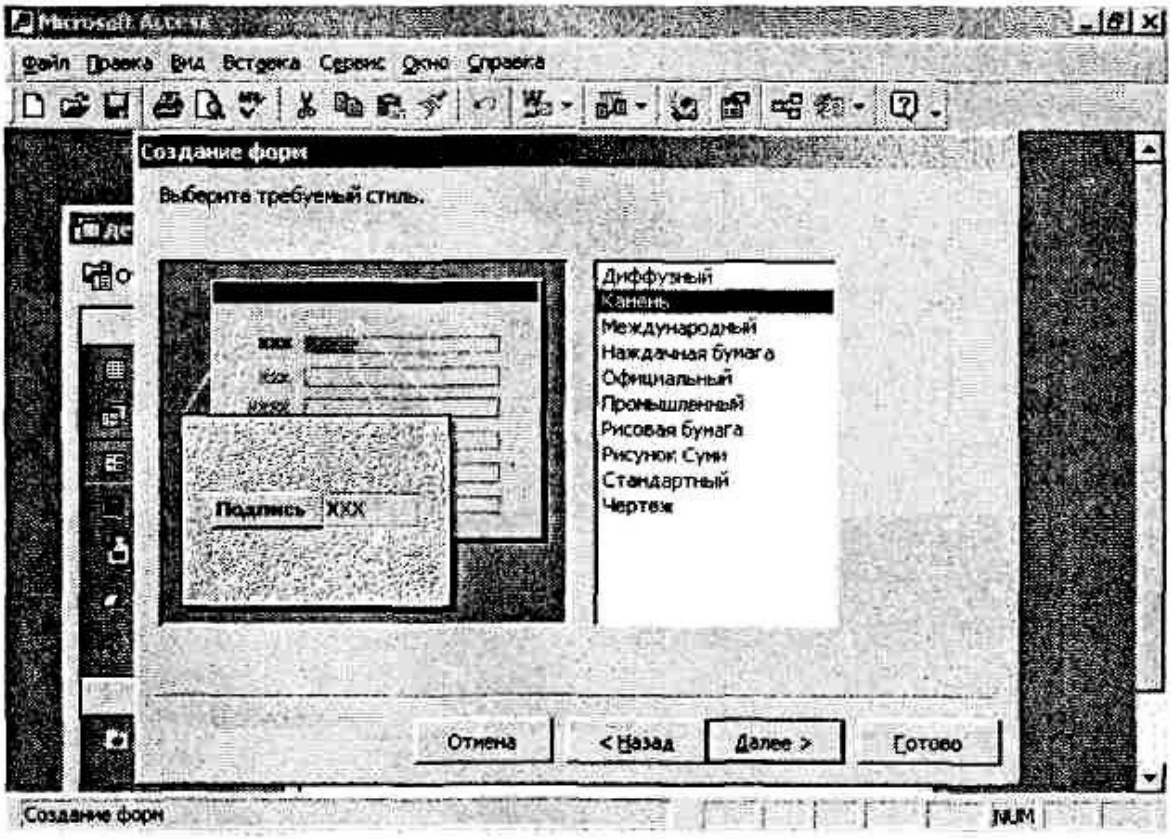


Рис. 8.10. Создание многотабличной формы – выбор стиля

При создании многотабличной формы система создаст описание двух форм: основной и подчиненной, поэтому при завершении создания формы следует задать соответственно имена для этих двух форм.

На рис. 8.11 представлен вид составной формы в режиме формы.

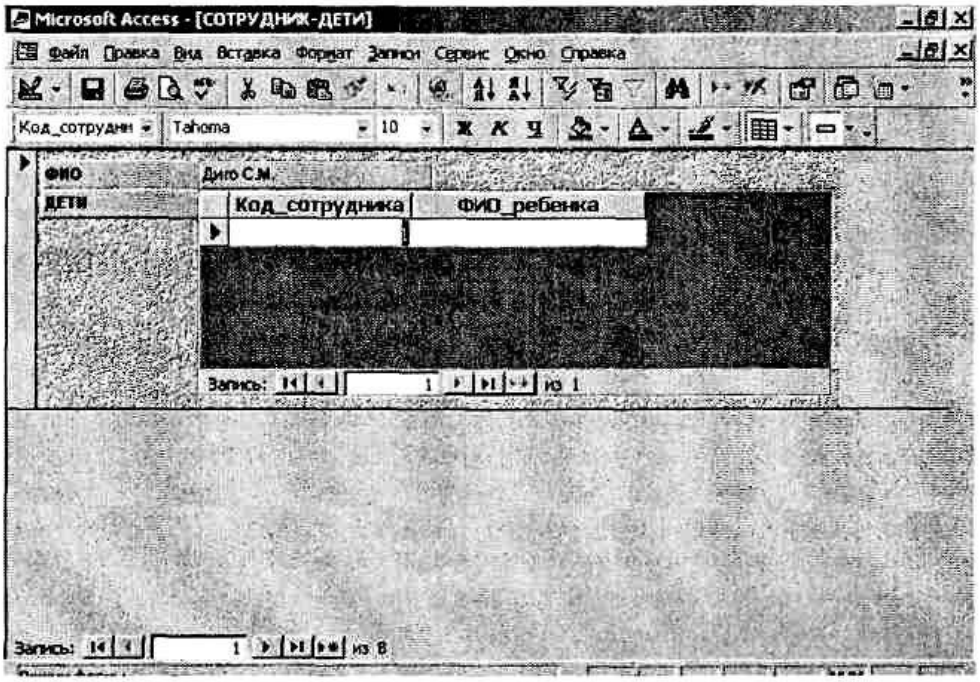


Рис. 8.11. Составная форма в режиме формы

При вводе данных в подчиненную форму код сотрудника вводится в подчиненную таблицу автоматически. Как видно, в подчиненной форме выводятся только записи, связанные с записью в главной форме.

При использовании составных экранных форм можно легко перемещаться по записям как подчиненной формы, так и главной формы. Для этого используется соответствующий набор кнопок перехода. Кроме того, для быстрого позиционирования на нужную запись можно воспользоваться возможностью поиска в БД. Для этого можно выбрать позицию меню **Правка/Найти** (или воспользоваться соответствующей кнопкой инструментального меню) и в появившемся окне ввести условие поиска.

Если при построении многотабличной формы сначала выбрать таблицу, находящуюся на стороне «многие» в отношении «1:М» (в нашем примере это таблица «Дети»), а потом таблицу, находящуюся на стороне «1» (в нашем примере это таблица «Сотрудник»), то многотабличная экранная форма также будет создана, но это будет совсем иная форма, чем та, что изображена на рис. 8.11. Это будет форма, в которой отображаются записи подчиненной таблицы (т.е. таблицы «Дети»), к которым присоединены поля из соответствующей записи основной таблицы. Никакой соподчиненности форм при этом не наблюдается. Создается только одно описание формы, а не два, как в предыдущем случае.

Другим способом создания многотабличной формы является создание запроса, отбирающего те поля из связанных таблиц, которые будут поме-

щаться в форму, и использование этого запроса в качестве источника для формы.

8.3.3. Корректировка формы в режиме Конструктор

Как отмечалось выше, форма, созданная Мастером, может быть скорректирована. Для этого необходимо одним из указанных ранее способов перейти в режим **Конструктор**. Экранная форма, полученная с помощью Мастера, в режиме **Конструктор** будет иметь вид, представленный на рис. 8.12.

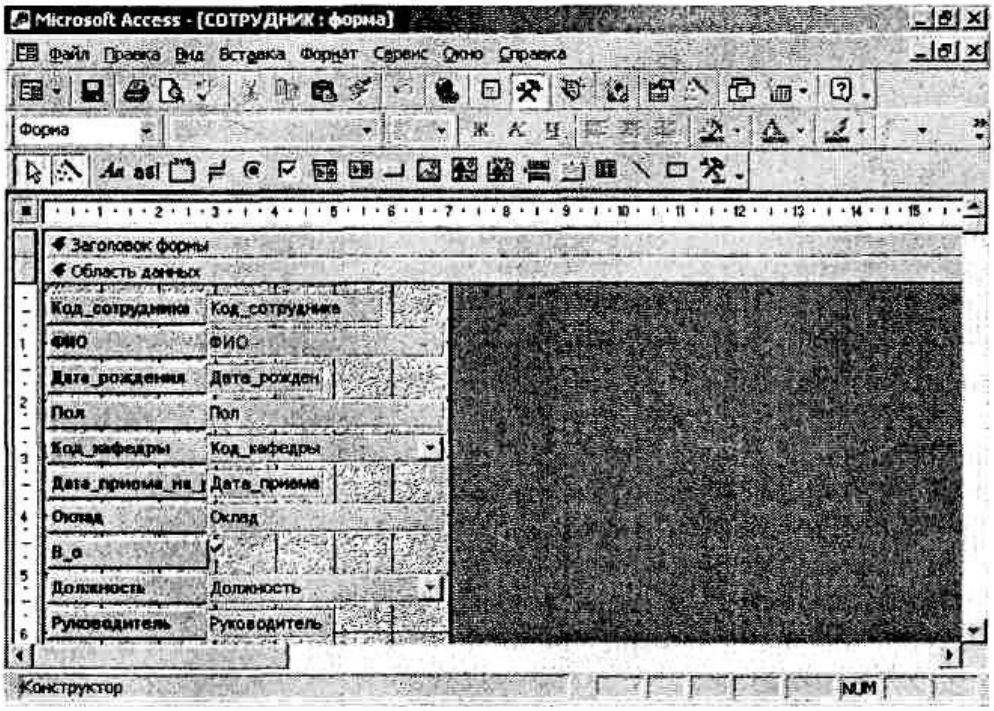


Рис. 8.12. Вид экранной формы в режиме **Конструктор**

В верхней части экрана видны три инструментальные панели: форматирования, конструктора форм и панель элементов. Первая из панелей - стандартная для всех офисных приложений MS. Многие элементы второй панели также привычны и понятны. Назначение некоторых специфичных кнопок будет пояснено ниже.

Как мы видим на рис. 8.12, все элементы полученной формы размещены в области данных. Область заголовка формы закрыта, но ее можно раскрыть и ввести в нее заголовок формы и другие данные, относящиеся ко всей форме. Поскольку назначение и способы работы с областями форм и отчетов одинаковы, а их использование в отчетах более актуально, то эти вопросы рассмотрены в главе 9, посвященной созданию отчетов.

Изменения, связанные с уже включенными в форму элементами управления

Изменения, вносимые в исходную форму, могут быть разнообразными. Прежде всего, существует возможность перемещения, изменения размеров, выравнивания, а также выполнения других действий с уже включенными в форму элементами управления. Для этого элемент (элементы), который нужно изменить, должен быть выделен. Для выделения элемента управления необходимо установить на него указатель и нажать кнопку мыши. Чтобы выделить несколько элементов управления, следует нажать клавишу [Shift] и, не отпуская ее, выделить все нужные элементы. Если выделяемые элементы находятся рядом и их не разделяют никакие элементы, которые не должны входить в выделяемую группу, то можно нажать левую клавишу мыши и, не отпуская ее, охватить появившимся контуром все те элементы, которые необходимо выделить. Выделенные элементы имеют характерные маркеры по углам элемента и по серединам его сторон.

Перемещение. Чтобы переместить выделенный элемент (элементы), нужно позиционироваться на нем мышью, добиться, чтобы указатель приобрел форму ладони, и, держа нажатой левую клавишу мыши, переместить элементы на требуемое место. В этом случае перемещаться будут все выделенные элементы. Если из пары «подпись»-«элемент управления» необходимо переместить что-то одно, то следует позиционироваться на левый верхний угол нужного элемента и добиться, чтобы указатель приобрел форму «указательного пальца»: в этом случае будет передвигаться только этот элемент.

Изменение размера. Чтобы изменить размер элемента, нужно позиционироваться на границе элемента таким образом, чтобы указатель принял форму двунаправленной стрелки. Выбор направления стрелок (вверх-вниз, вправо-влево, по диагонали) зависит от того, как вы хотите изменить размер элемента.

Выравнивание. Чтобы выровнять выделенные элементы управления, в меню **Формат** следует выбрать команду **Выровнять**, а затем в появившемся списке выбрать способ выравнивания.

Удаление. Чтобы удалить выделенные элементы управления, необходимо нажать клавишу [Del].

Включение новых элементов в форму

Если первоначально были включены не все поля из таблицы (или случайно удален нужный элемент), то нетрудно добавить в форму поля из таблицы, являющейся источником данных формы. Для этого следует щелкнуть по кнопке **Список полей** на панели инструментов. Из появившегося ниспадающего списка необходимо выбрать имя нужного поля, позиционировавшись на нем, нажать левую клавишу мыши и, не отпуская ее, переместить элемент в нужное место формы.

Более сложным является вариант, когда для этих целей используется кнопка **Поле** на панели элементов, а потом у вставленного в результате этого действия свободного элемента меняется соответствующим образом свойство «Данные», а у его надписи - свойство «Подпись». Но такой способ лучше использовать только в случае, когда иной путь невозможен, например при выводе в форму вычисляемого поля. При создании вычисляемого поля в свойство «Данные» необходимо ввести выражение для вычисления значения этого поля.

Вычисляемые поля могут вводиться не только в те формы, которые используются для вывода информации, но и в те, которые используются для ввода данных в базу данных. Например, при вводе данных в таблицу «Сотрудник» можно на экран вывести вычисляемое поле «Возраст»: это поле не будет храниться в таблице (в нее будет вводиться только «Дата рождения»), а на экран автоматически при вводе даты рождения будет выводиться возраст, что удобно, например, для контроля вводимых данных.

Чтобы ввести в форму текст, необходимо щелкнуть по кнопке **Надпись** на панели элементов и, не отпуская кнопку мыши, переместиться в то место в форме, куда следует поместить текст, после чего ввести нужный текст и нажать клавишу [Enter].

Кроме текста и полей в форму могут быть включены линии, квадраты, рисунки.

Изменение типа элемента управления

В экранной форме могут использоваться разные элементы управления, в том числе список, поле со списком, которые широко используются при создании экранных форм.

Если при создании таблицы поля были созданы как поля подстановки, то в форме, полученной в результате использования Мастера, этим полям будут соответствовать поля со списком.

Если поля были созданы как обычно, а вы хотите в форме использовать, например, поле со списком, то достичь желаемого результата можно разными способами.

Во-первых, можно изменить тип элемента управления в форме. Для этого в режиме **Конструктор** необходимо выделить соответствующий элемент формы, щелкнуть правой клавишей мыши и в появившемся контекстном меню выбрать позицию «**Преобразовать элемент в...**» (рис.8.13).

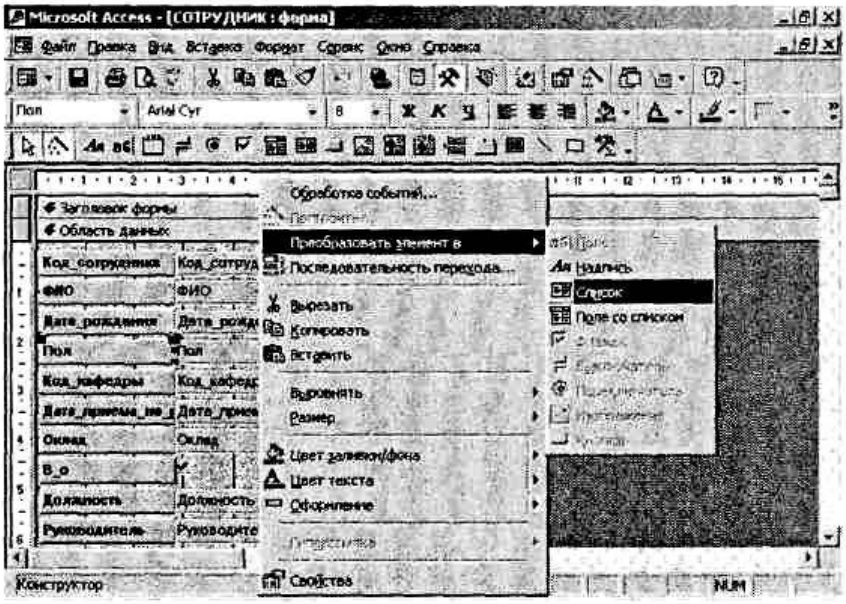


Рис. 8.13. Преобразование элемента

В табл. 8.1 приведены возможные варианты преобразования. Доступные преобразования будут зависеть от того, какой тип имеет выбранный элемент.

Но в некоторых вариантах преобразования необходимо выполнить дополнительные шаги для того, чтобы достичь желаемого результата. Так, например, если обычное поле преобразовать в «список» или «поле со списком», то автоматически список значений или связь с полем подстановки не появится. Поэтому следует соответствующим образом изменить свойства элемента. Для этого нужно позиционироваться на нужном элементе и щелкнуть по кнопке **Свойства**. Например, если мы хотим сделать элемент «Пол» списком с фиксированным набором значений «м» и «ж», то в поле «Тип источника строк» нужно выбрать «**Список значений**», а в качестве источника строк через точку с запятой указать «м» и «ж» (рис. 8.14).

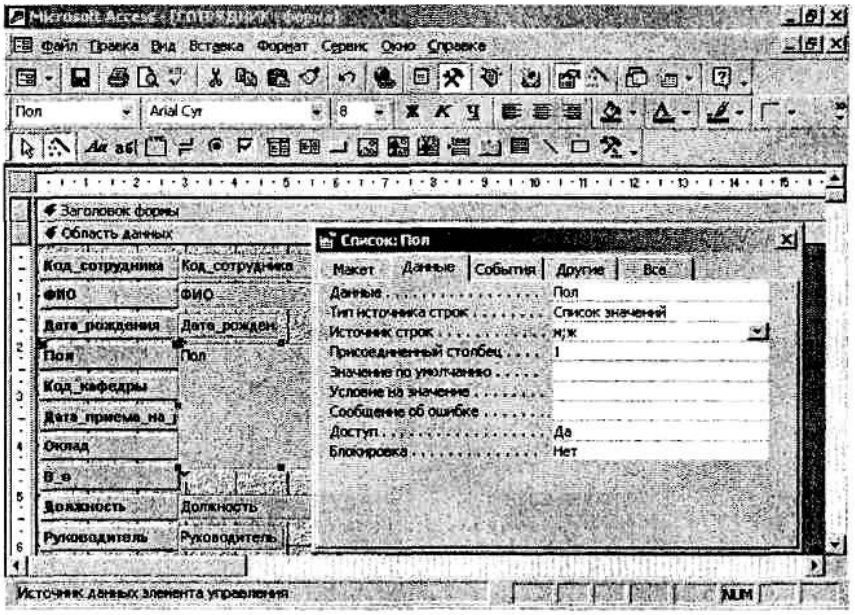


Рис. 8.14. Изменение свойств элемента при преобразовании типа элемента в «список»

Таблица 8.1

Элемент управления	Поле	Надпись	Список	Поле со списком	Флажок	Выключатель	Переключатель	Изображение	Кнопка
Поле (не логическое)		+	+	+					
Надпись	+								
Список	+			+					
Поле со списком	+		+						
Поле (логическое): флажок выключатель переключатель					+ + +	+ + +	+ + +		
Изображение									
Кнопка									

Если элемент типа «поле» преобразуется в тип «поле со списком», то изменение свойств будет еще сложнее. Так, например, если мы хотим элемент, соответствующий полю «Код_кафедры», преобразовать в поле со списком, то вид «источника строк» будет «таблица/ запрос»; в качестве источника строк следует выбрать таблицу «Кафедра», после чего щелкнуть мышью по строке «Источник строк» и затем - по кнопке с многоточием, чтобы вызвать построитель запросов (рис. 8.15).

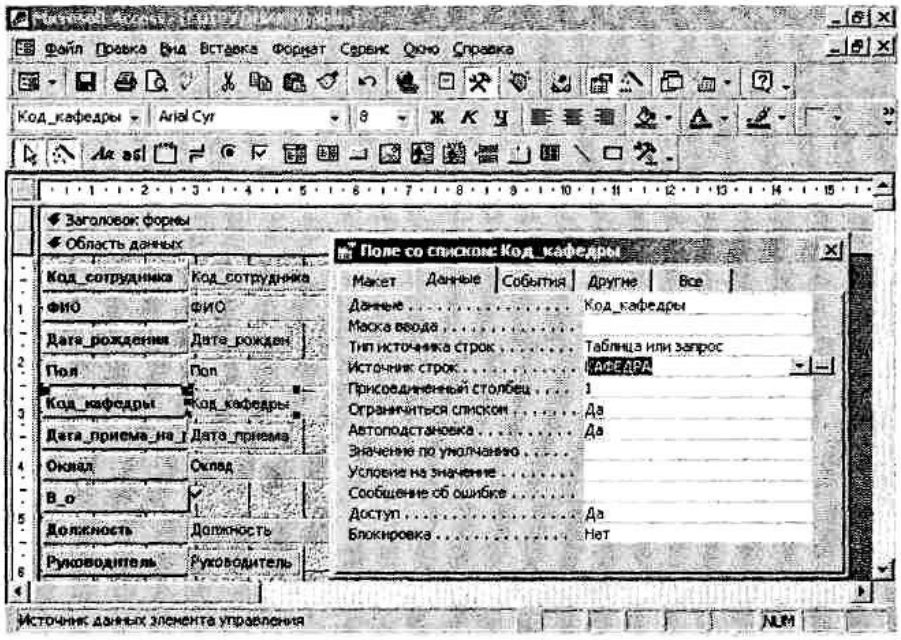


Рис. 8.15. Изменение свойств элемента при преобразовании типа элемента в «поле со списком»

Создаваемый запрос будет зависеть от того, только столбец подстановки будет выводиться на экран или еще и поясняющий его столбец, т.е. в нашем примере столбец «Код_кафедры» и «Наименование_кафедры_полное». В этом случае запрос примет вид, представленный на рис. 8.16.

Кроме того, может потребоваться изменение свойств «Число столбцов» и «Ширина столбцов».

Как видим, при отсутствии навыков такое преобразование является не совсем тривиальным.

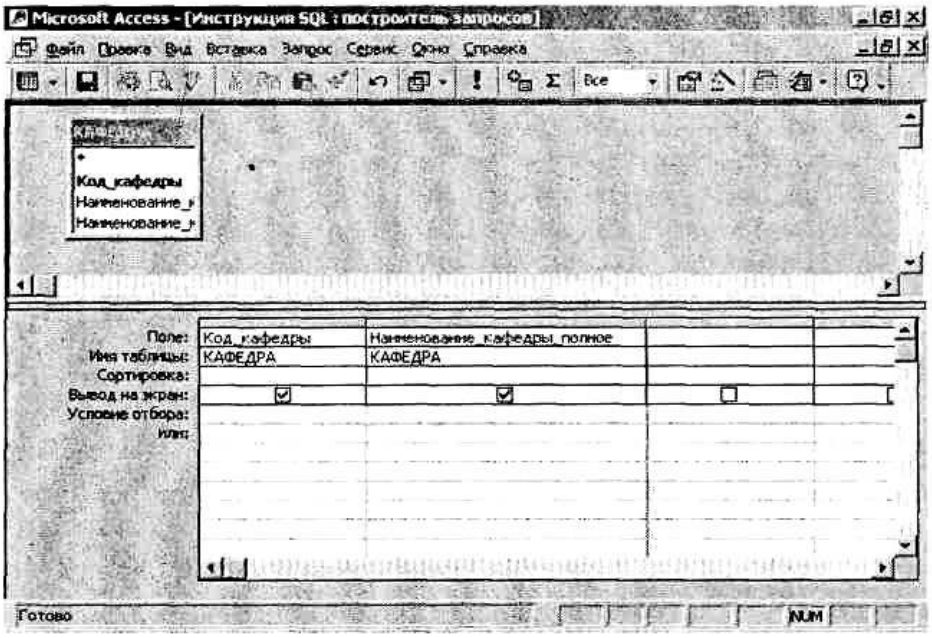


Рис. 8.16. Вид окна **Инструкция SQL**

Можно воспользоваться и другим вариантом «преобразования» типа элемента, а именно удалить элемент из формы и создать его заново, выбрав на панели элементов элемент нужного типа.

Если вы работаете с версией Access, позволяющей при создании таблицы определять поле подстановки, то лучше воспользоваться этой возможностью.

Создание форм, состоящих из нескольких страниц

Существует несколько причин, по которым бывает необходимо/ целесообразно разнести данные, размещаемые в экранной форме, по нескольким страницам. Это может быть в случае, когда элементов в форме много и размещение всех их на одном экране слишком загромождает его; либо может быть вызвано желанием сгруппировать поля и показывать каждую группу отдельно и др. Для достижения этих целей можно, воспользовавшись элементом управления **Разрыв страницы**, указать, в каких местах должен быть переход на следующую страницу, или создать форму с несколькими вкладками (рис. 8.17).

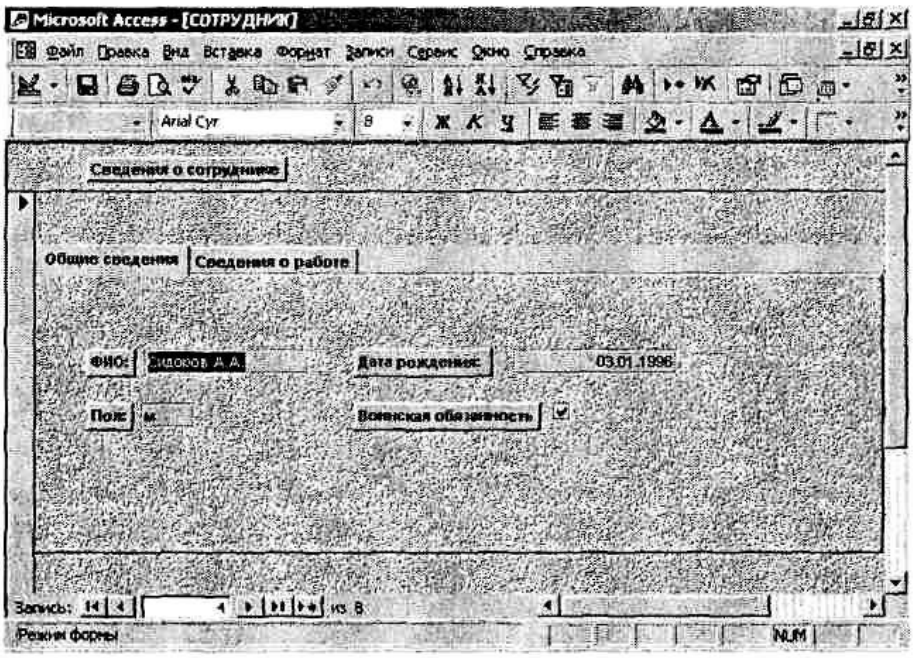


Рис. 8.17. Экранная форма с вкладками. Режим формы

Элемент управления **Разрыв страницы** используется для указания горизонтальных разрывов между элементами управления в форме. Для перехода к странице, находящейся над или под указанным разрывом, используются клавиши [Page up] или [Page down].

Для того чтобы создать форму с несколькими вкладками, можно воспользоваться элементом управления **Набор вкладок**.

При выборе этой возможности в форме создаются две вкладки. В каждую из них можно переместить необходимые элементы. Если требуется создать большее число вкладок, то следует, находясь в зоне вкладок, нажать правую клавишу мыши и в появившемся контекстном меню выбрать позицию **Добавить вкладку** (рис. 8.18). Для включения элементов нужно выбрать необходимую вкладку и щелкнуть по кнопке **Список полей** на панели инструментов. Из появившегося ниспадающего списка следует выбрать имя нужного поля, позиционировавшись на нем, нажать левую клавишу мыши и, не отпуская ее, переместить элемент в нужное место вкладки.

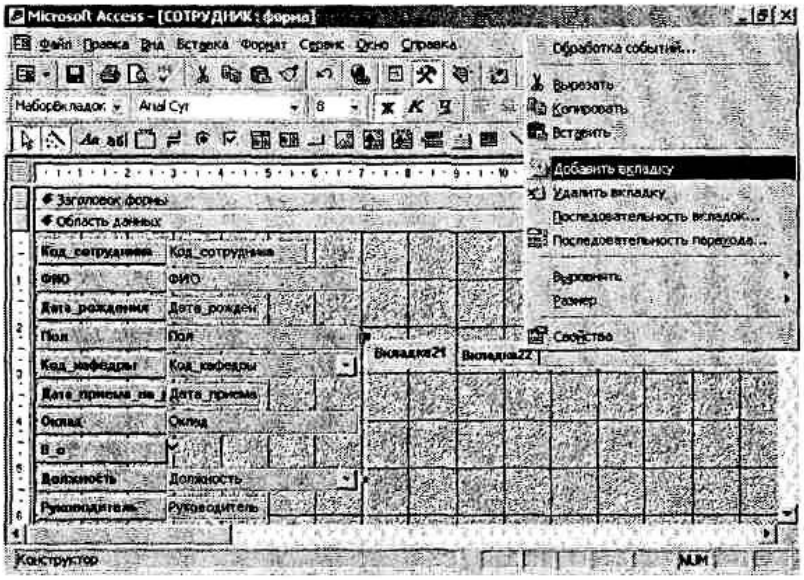


Рис. 8.18. *Добавление вкладок*

Для того чтобы изменить название вкладки, необходимо воспользоваться правой кнопкой мыши, в появившемся контекстном меню выбрать позицию Свойства и в свойстве «Имя» записать требуемое название (рис. 8.19).

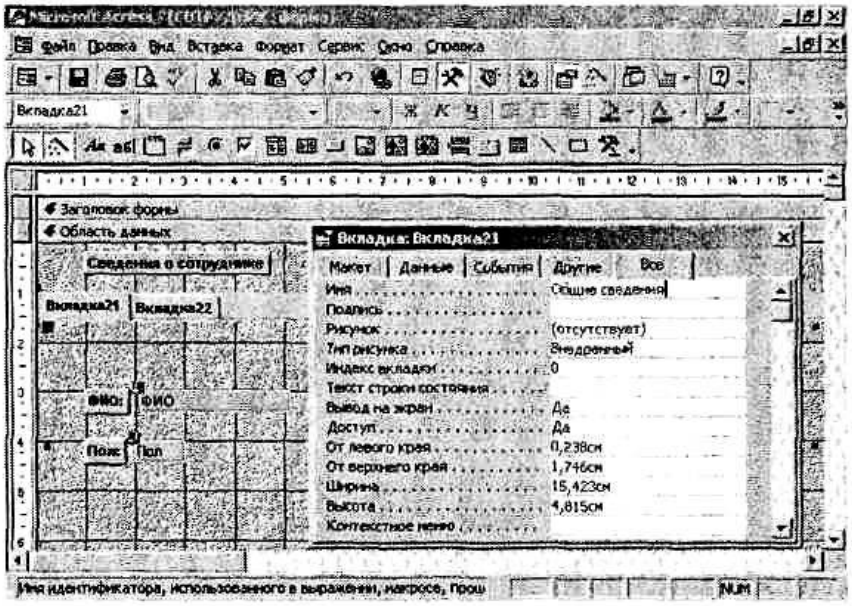


Рис. 8.19. Изменение названия вкладки

Последовательность обхода полей

Последовательность обхода полей при работе с формой может отличаться от их расположения на экране. Для установления этой последовательности можно, позиционировавшись на заголовок окна формы в режиме **Конструктор**, нажать на правую клавишу мыши. При этом появится всплывающее окно (рис. 8.20), в котором следует выбрать позицию **Последовательность перехода....**

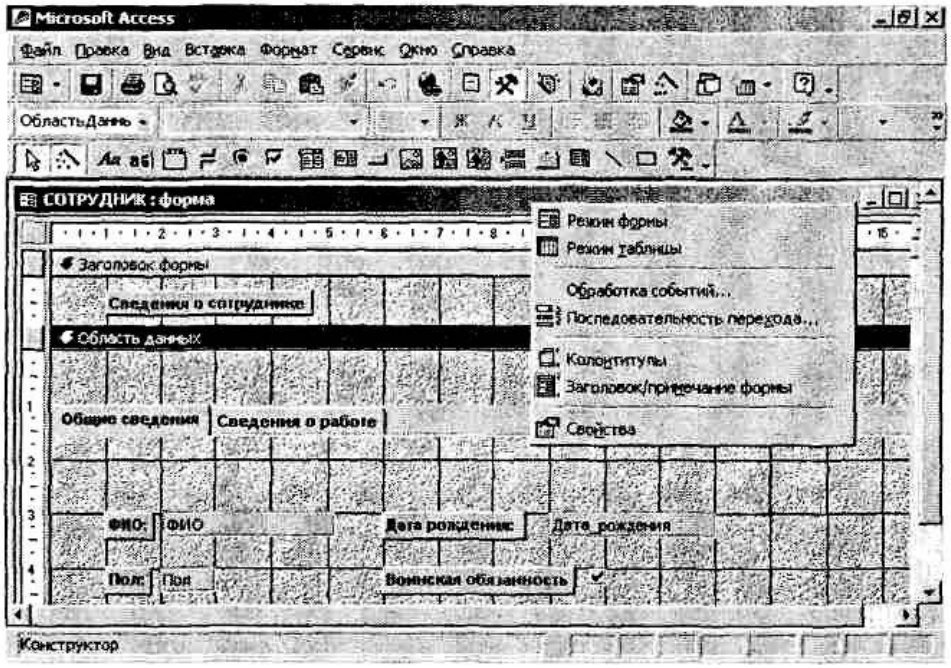


Рис. 8.20. Задание последовательности обхода полей (экран 1)

После этого на экране появится окно (рис. 8.21), в котором перечислены поля, включенные в форму (если форма содержит несколько вкладок, то будут выводиться только те элементы, которые включены в активную вкладку).

Затем поля формы следует разместить в нужной последовательности. Если речь идет о формах, используемых для ввода данных, то желательно, чтобы последовательность обхода полей совпадала с расположением соответствующих реквизитов в документе, который используется в качестве источника данных.

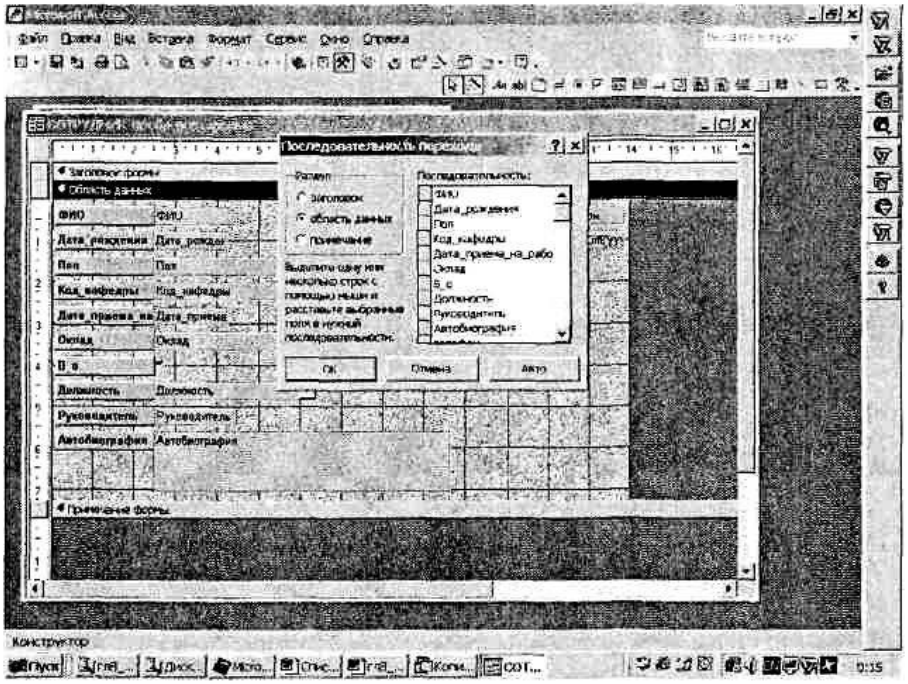


Рис. 8.21. Задание последовательности обхода полей (экран 2)

Свойства формы

Кроме свойств, присущих каждому отдельному элементу формы, имеются свойства, относящиеся ко всей форме. Их можно с успехом использовать для создания дополнительных удобств при работе с формой, для обеспечения целостности базы данных и других целей. Для того чтобы посмотреть/скорректировать свойства формы, надо открыть форму в режиме **Конструктор** и двойным нажатием кнопки мыши на области выделения формы открыть окно свойств формы. Перечень свойств формы обширен. Рассмотрим некоторые из них. Так, на вкладке **Данные** (рис. 8.22) имеется свойство «Ввод данных». Если выбрать для него значение **Да**, то можно создать форму, использующуюся только для ввода данных. В этом случае в форме будет высвечиваться одна пустая запись, в которую можно вводить новые данные. Если нужно, чтобы выводились все записи, значение этого свойства должно быть **Нет**.

Можно, напротив, создать форму только для просмотра, запретив все корректировки.

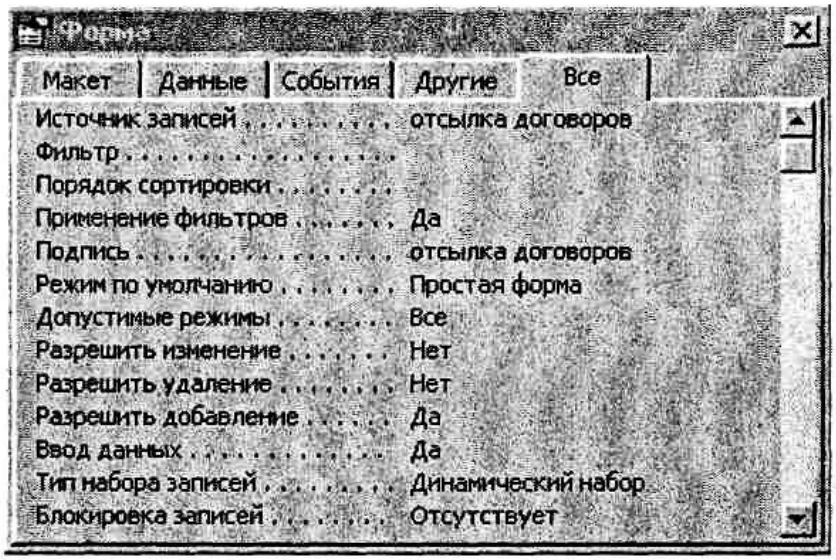


Рис. 8.22. Свойства формы

Задание ограничений целостности при создании форм

Ограничения целостности можно задавать не только при описании базы данных, но и при создании экранных форм, предназначенных для ввода данных. Для элемента управления, соответствующего полю, можно задать свойство «Условие на значение» (ValidationRule). Условие для элемента управления не отменяет условие для поля, которое было задано при создании таблицы, поэтому при задании ограничений целостности необходимо следить, чтобы они не противоречили друг другу. Следует иметь в виду, что ограничения целостности, заданные для поля при описании таблиц, действуют все-

гда, а ограничения, заданные при создании экранной формы, - только при использовании этой формы для ввода информации.

Для элемента управления, в который осуществляется ввод данных, можно создать маску ввода и условие на значение.

Чтобы предотвратить ввод данных, можно заблокировать соответствующий элемент управления, задав значение свойства «Доступ» - «Нет». В этом случае содержимое поля будет видно на экране, но изменить его будет нельзя.

Добавление кнопок в форму

Важным элементом формы является **Кнопка**. К кнопкам привязываются определенные действия, которые выполняются при щелчке по этой кнопке.

Для того чтобы включить кнопку в форму, можно воспользоваться элементом **Кнопка** на панели элементов. Для этого надо выделить этот элемент, щелкнув по нему левой кнопкой мыши, после чего позиционироваться на том месте формы, где предполагается расположить кнопку, и опять нажать левую кнопку мыши - кнопка появится на экране.

После этого следует, позиционировавшись на этой кнопке, нажать правую кнопку мыши и в появившемся контекстном меню выбрать позицию **Свойства**. Кнопка имеет большой набор свойств (рис. 8.23). Набор этих свойств так велик, что весь не умещается на экран. Рассмотрим некоторые из этих свойств.

В свойстве кнопки «Подпись» можно задать подпись, которая появится на кнопке. Например, если в этом свойстве записать «закрытие формы», то кнопка примет вид, изображенный на рис. 8.24.

Вместо подписи на кнопке можно поместить рисунок. Для этого необходимо позиционироваться на строке, соответствующей свойству «**Рисунок**», щелкнуть по кнопке с многоточием и в появившемся окне Рисунки (рис. 8.25) выбрать подходящий рисунок из списка или найти его на диске (воспользовавшись кнопкой **Обзор** ...).

Если выбрать рисунок **Закрытие формы**, то кнопка будет иметь вид, представленный на рис. 8. 26.

Далее необходимо поставить в соответствие кнопке действие, которое будет выполняться при щелчке по этой кнопке. Для этого можно, например, выбрать свойство «Нажатие кнопки». Щелкнув по кнопке с многоточием в этой строке, попадаем в окно **Построитель** (рис. 8.27). Для своей кнопки выберем позицию **Макросы**, затем - макрокоманду **Закрыть**, указав при этом тип и имя объекта (рис. 8.28).

Для создания кнопки можно воспользоваться и Мастером кнопок. Для этого кнопка **Мастер** на панели элементов должна быть нажата. В этом случае при использовании элемента **Кнопка** откроется окно **Создание кнопок** (рис. 8.29). В появившемся окне можно выделить несколько зон. Прежде всего это два списка: *Категории* (фактически означающие группы действий) и Дейст-

вия, относящиеся к каждой из этих категорий. Слева высвечивается рисунок, соответствующий выбранному действию.

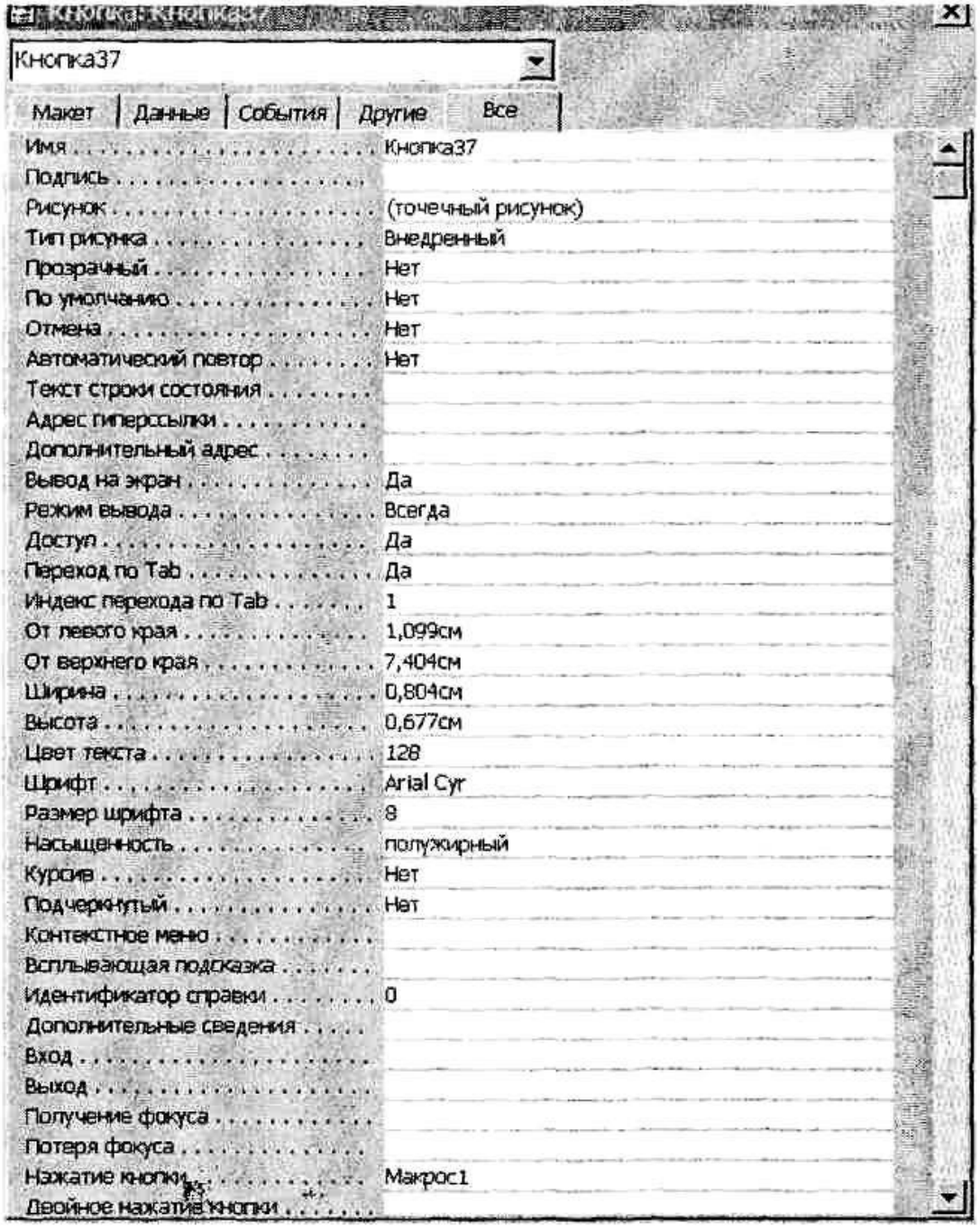


Рис. 8.23. Свойства элемента **Кнопка**

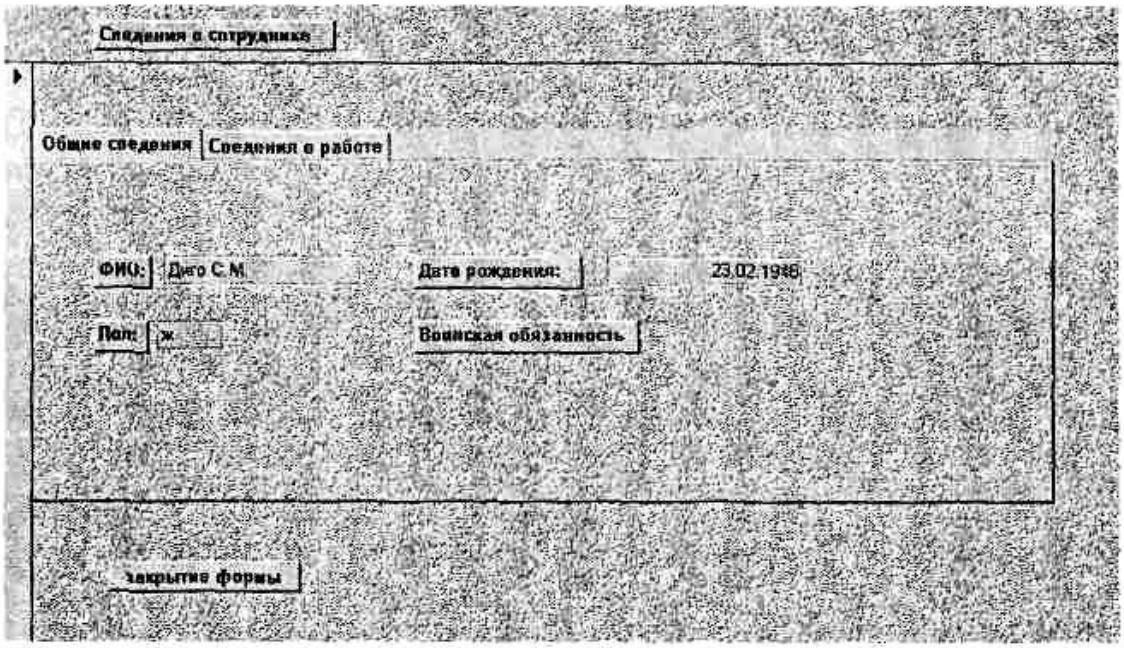


Рис. 8.24. Форма с добавленной кнопкой **Закрытие формы**

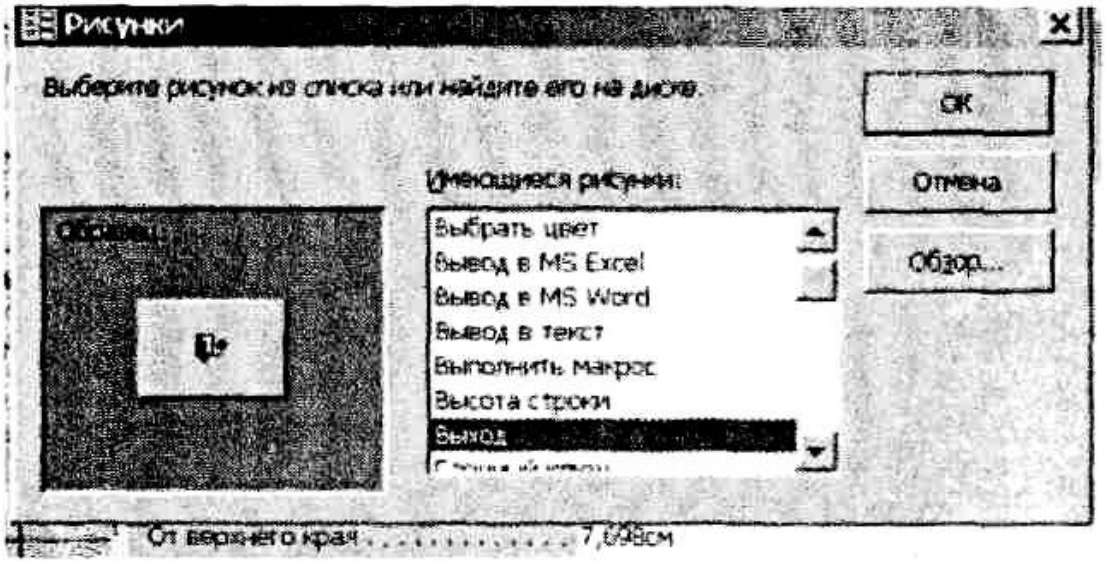


Рис. 8.25. Выбор рисунка для кнопки

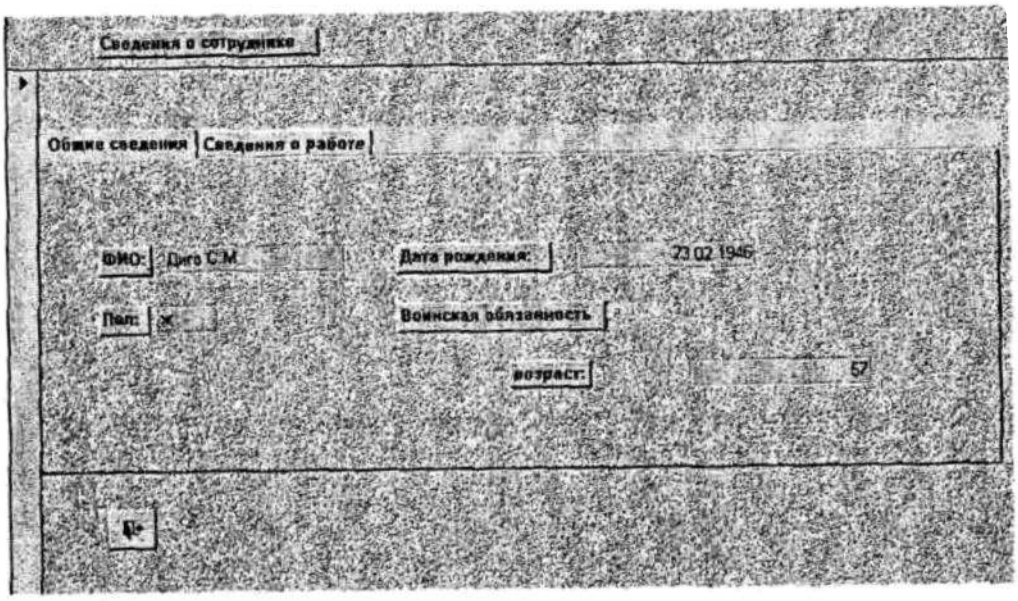


Рис. 8.26. Форма с кнопкой **Закрытие формы**, использующей рисунок

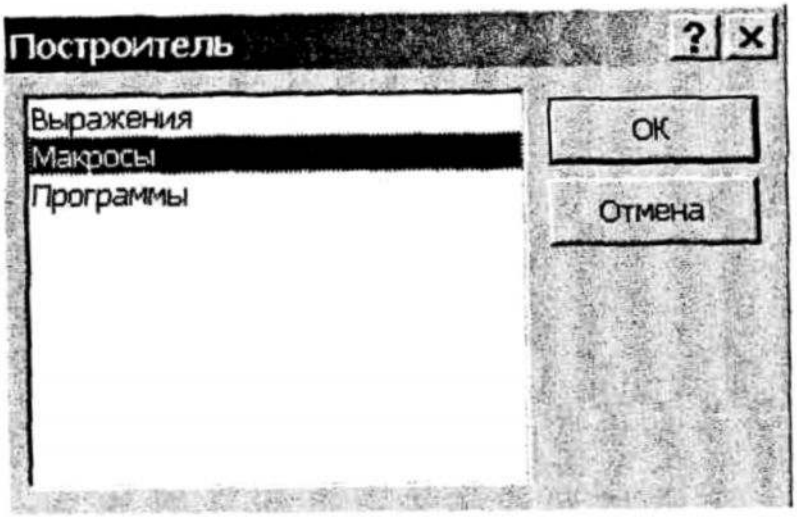


Рис. 8.27. Вид окна **Построитель**

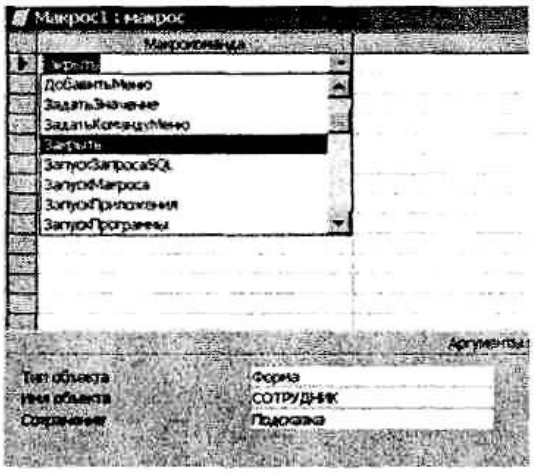


Рис. 8.28. Выбор макрокоманды

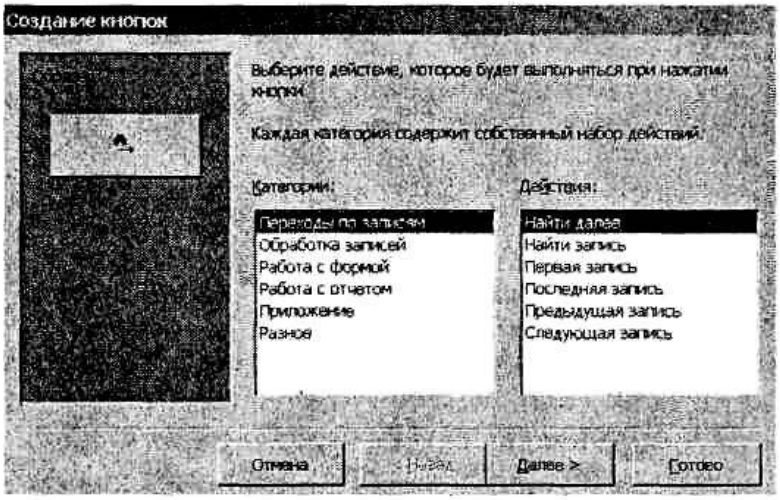


Рис. 8.29. Вид окна **Создание кнопок**

На следующем шаге дается возможность уточнить выбор рисунка или задать надпись на кнопке, после чего следует задать имя кнопки.

8.3.4. Кнопочная форма

Кнопочная форма - это форма, открывающая другие формы или отчеты базы данных. В англоязычном варианте используется название «switchboard», что в дословном переводе означает «коммутационная панель». Такое определение больше отражает суть создаваемого объекта. Кнопочную форму можно создать, используя описанные выше приемы включения кнопок в форму, а можно воспользоваться специальной служебной программой **Диспетчер кнопочных форм**. Для того чтобы запустить эту программу, в меню **Сервис**

выберите команду **Служебные программы**, затем - **Диспетчер кнопочных форм.**

Если кнопочная форма создается в данном приложении впервые, то выдается сообщение «Не удается найти кнопочную форму в этой базе данных. Создать кнопочную форму?», на которое следует ответить Да, после чего появится окно **Диспетчер кнопочных форм** (рис. 8.30).

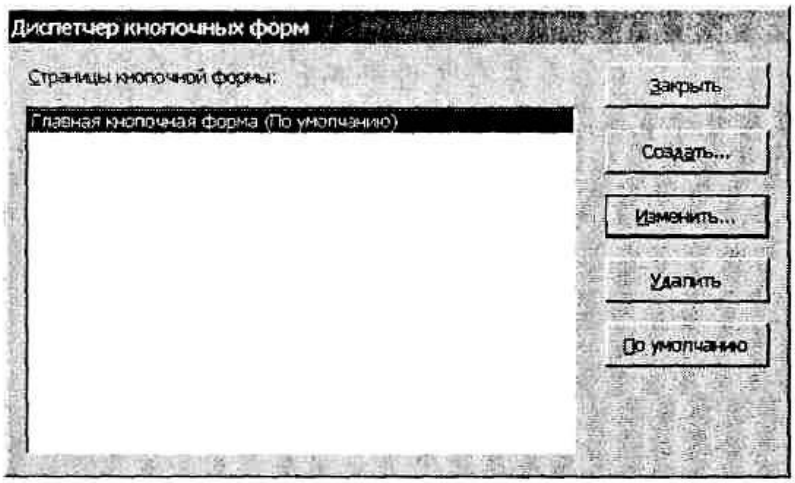


Рис. 8.30. Вид окна **Диспетчер кнопочных форм**

Первоначально кнопочная форма не содержит никаких элементов. Для того чтобы продолжить работу по созданию кнопочной формы, необходимо сначала концептуально определить, как будет выглядеть наша кнопочная форма. Если это будет одноуровневая форма, которая включает кнопки, соответствующие конечным действиям, то следует щелкнуть по кнопке **Изменить**... и внести в нее необходимые элементы. Если создаваемый интерфейс будет многоуровневым, при котором выполняемые функции будут сгруппированы и каждая группа кнопок будет размещаться на отдельной странице кнопочной формы, то необходимо создать эти дополнительные страницы с помощью кнопки Создать.... Предположим, что мы хотим сгруппировать выполняемые функции и создать три кнопочные формы второго уровня: «Ввод данных», «Корректировка данных» и «Отчеты». Создадим эти новые страницы, воспользовавшись кнопкой Создать... -появится окно Создание (рис. 8.31), в котором следует задать имя новой страницы кнопочной формы.

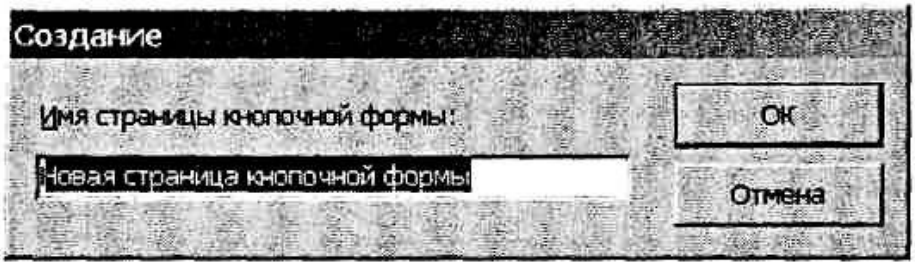


Рис. 8.31. Вид окна создания новой страницы кнопочной формы

После создания трех требуемых нам страниц диалоговое окно Диспетчер кнопочных форм приобрело вид, представленный на рис. 8.32.

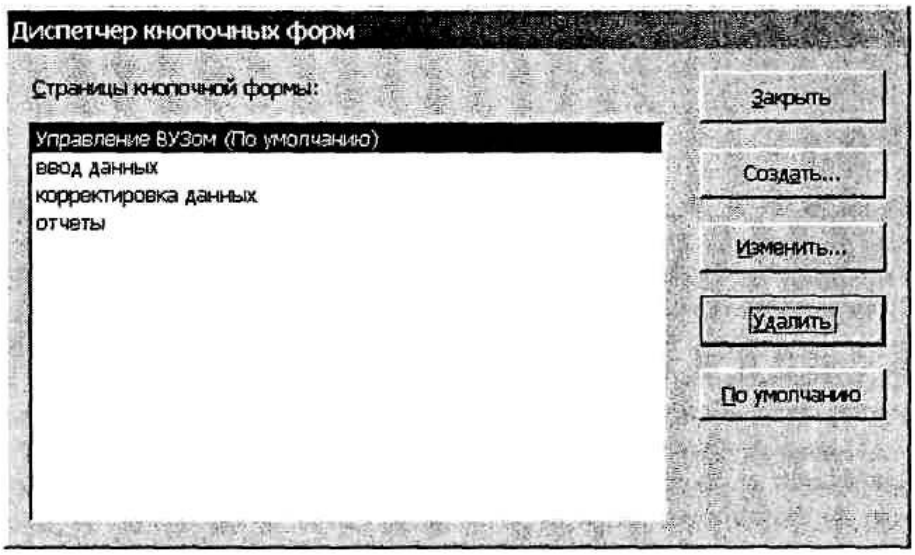


Рис. 8.32. Вид окна **Диспетчер кнопочных форм** после описания новых страниц

Определим теперь элементы для каждой из страниц кнопочной формы. Для этого позиционируемся на имени соответствующей страницы и щелкнем по кнопке «**Изменить**...». В диалоговом окне **Изменение страницы кнопочной формы** (рис. 8.33) следует для каждой страницы внести необходимые для нее элементы. При первичном открытии этого окна (пока не создано ни одного элемента) активной является только кнопка **Создать**... При щелчке по этой кнопке появляется диалоговое окно **Изменение элемента кнопочной формы** (рис. 8.34). В поле «Текст» следует ввести имя создаваемой кнопки, например, для получения отчета на выплату зарплаты, можно так и назвать кнопку.

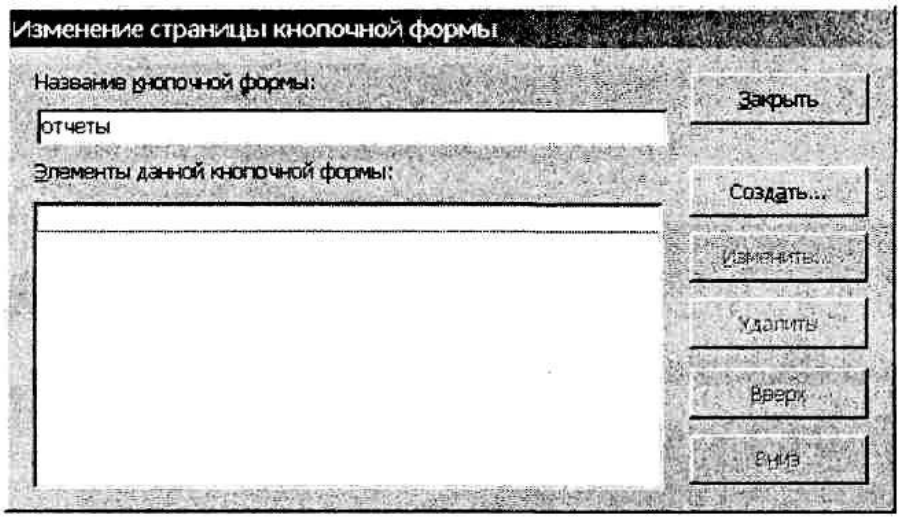


Рис. 8.33. Вид окна **Изменение страницы кнопочной формы**

Список доступных команд высвечивается при нажатии на стрелку в поле «Команда» (рис. 8.35). Если мы создаем элементы для страницы «Отчеты», то необходимо выбрать команду **Открыть отчет**; если создаем элементы для страницы «Ввод данных» - **Открыть форму для добавления** и т.д.

В нашем примере мы демонстрируем описание страницы кнопочной формы «Отчеты». Поэтому была выбрана команда **Открыть отчет**. В связи с этим последнее поле диалогового окна **Изменение элемента кнопочной формы** стало называться **Отчет**, и при нажатии на стрелку в этом поле высвечивается список отчетов, включенных в базу данных (рис. 8.36). Из этого списка следует выбрать нужный (в нашем случае это отчет «Ведомость з/п»), далее - повторить шаг **Создать элемент кнопочной формы** столько раз, сколько элементов требуется поместить на данную страницу. Acces накладывает ограничение на число кнопок, которые можно разместить на одной странице, - не более восьми.

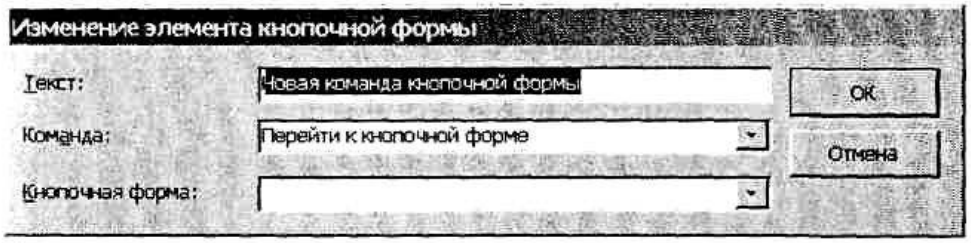


Рис. 8.34. Вид окна **Изменение элемента кнопочной формы**

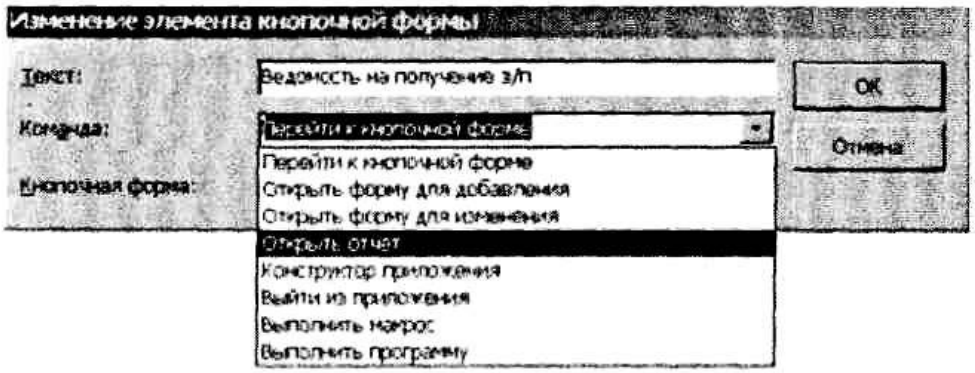


Рис. 8.35. Вид окна **Изменение элемента кнопочной формы** со списком возможных команд

Если необходимое число действий превосходит это ограничение, то следует либо пересмотреть разбиение формы на страницы и сгруппировать действия так, чтобы на каждой странице это число не превышало восьми, либо после завершения формирования кнопочной формы с помощью утилиты «Диспетчер кнопочных форм» перейти в режим **Конструктор** и допроектировать форму вручную.

Надо не забыть, что мы описали каждую из подчиненных страниц главной кнопочной формы, но не подключили их к главной форме. Для того чтобы нужная страница открывалась из главной формы, необходимо выполнить

следующие действия: позиционироваться на позиции главной кнопочной формы (мы ее назвали «Управление вузом»), щелкнуть по кнопке **Изменить**..., далее, позиционировавшись поочередно на каждом элементе этой формы, также щелкнуть по кнопке **Изменить**... и внести соответствующие изменения.

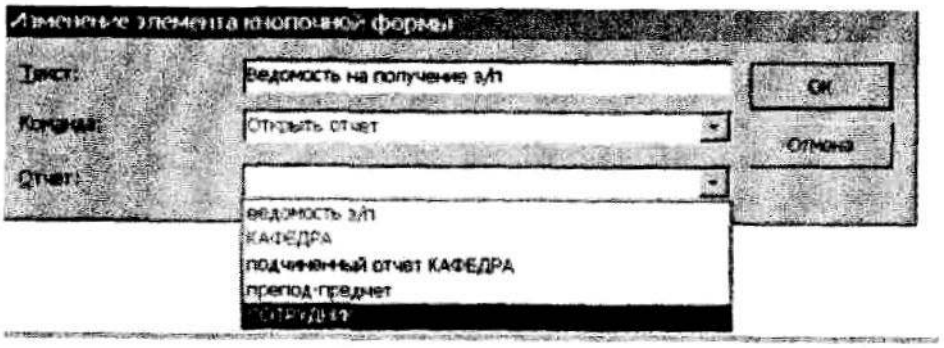


Рис. 8.36. Вид окна **Изменение элемента кнопочной формы** со списком доступных отчетов

Страница кнопочной формы не обязательно должна содержать однородные элементы, как это было в нашем примере. Например кроме открытия кнопочных форм второго уровня введем в главную форму еще кнопку для закрытия главной кнопочной формы и выхода из приложения. Для этого, находясь в окне **Диспетчер кнопочных форм** и позиционировавшись на строке «Управление вузом» необходимо щелкнуть по кнопке **Изменить**... и в появившемся окне **Изменение элемента кнопочной формы** щелкнуть по кнопке **Создать…**, задать имя этой кнопки, а в качестве команды выбрать **Выйти из приложения** (рис. 8.37).

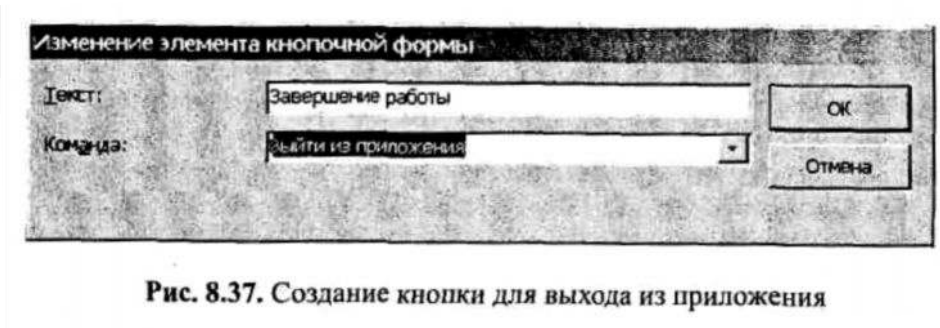


Рис. 8.37. Создание кнопки для выхода из приложения

Для того чтобы из каждой страницы второго уровня можно было вернуться к главной странице, в них создается кнопка возврата в главное меню (рис. 8.38).

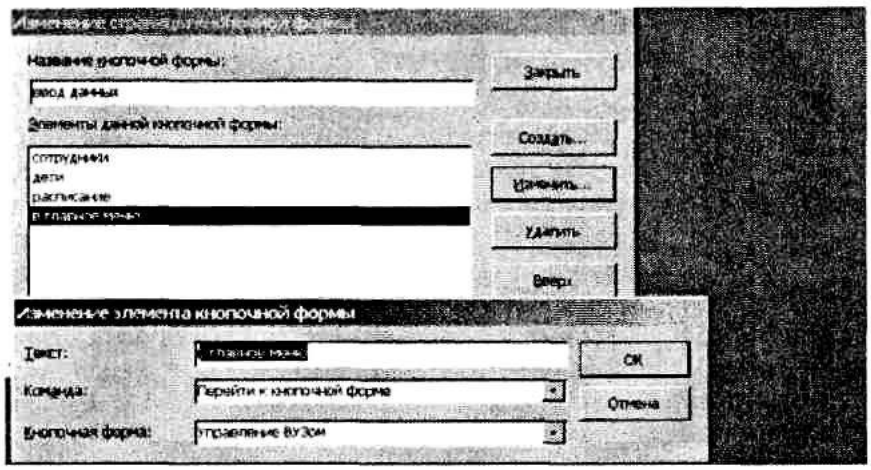


Рис. 8.38. Создание кнопки перехода в Главное меню

Когда создается кнопочная форма с помощью «Диспетчера кнопочных форм», то система кроме создании самой формы создает вспомогательную таблицу базы данных, которая называется Switchboard Items. Эта таблица содержит метаинформацию, а именно - описание каждого элемента кнопочной формы. В нашем примере была сформирована таблица следующего содержания (табл. 8.2).

Таблица 8.2

Switchboard Items : таблица

SwitchboardID	ItemNumber	ItemText	Command	Argument
1	0	Управление ВУЗом		По умолчанию
1	1	ввод данных	1	2
1	2	корректировка данны	1	3
1	3	отчеты	1	4
1	4	Завершение работы	6	
2	0	ввод данных	0	
2	1	сотрудники	2	СОТРУДНИК
2	2	дети	2	СОТРУДНИК-Д
2	3	расписание	2	ввод расписан
2	4	в главное меню	1	1
3	0	корректировка данны	0	
4	0	отчеты	0	
4	1	Ведомость на получе	4	ведомость з/п
4	2	состав кафедр	4	КАФЕДРА
4	3	сведения о сотрудни	4	СОТРУДНИК
	0			

8.3.5. Возможные случаи возникновения ошибок

Если в таблице, которая являлась источником данных для формы, будет переименовано или удалено поле, включенное в форму, то при выводе формы в режиме формы будет в соответствующем ему окошке выводиться строка «#Имя?». Это является примером нарушения целостности базы данных, когда был изменен один элемент базы данных и не изменен связанный с ним

другой элемент (в данном случае - поле таблицы базы данных и базирующийся на нем элемент экранной формы). Если поле было переименовано, то для соответствующего элемента формы необходимо скорректировать свойство «Источник данных». Если поле было удалено из таблицы, то соответствовавший ему элемент нужно удалить из формы.

Все вышесказанное говорит о том, что желательно при первоначальном проектировании и создании таблиц максимально продумывать проектные решения и с осторожностью подходить к дальнейшей корректировке структуры таблиц базы данных.

При создании форм ввода следует соотносить состав полей, включенных в форму, с составом полей и, главное, с ограничениями целостности, которые наложены на эти поля при описании таблиц баз данных. Так, если в форму не будут введены поля, на которые наложены ограничения на значения, то новые записи в таблицу не смогут быть введены из-за несоблюдения этих ограничений.

8.3.6. Открытие формы в режиме сводной таблицы или в режиме диаграммы

Формы можно также открывать в режиме сводной таблицы или в режиме диаграммы для анализа данных*. В этих режимах пользователи могут динамически изменять макет формы для изменения способа представления данных. Существует возможность упорядочивать заголовки строк и столбцов, а также применять фильтры к полям. При каждом изменении макета сводная форма немедленно выполняет вычисления заново в соответствии с новым расположением данных.

В режиме сводной таблицы имеется возможность просматривать исходные данные или сводные данные, упорядочивая поля в областях фильтра, строк, столбцов и данных.

В режиме сводной диаграммы можно визуально представлять данные, выбрав тип диаграммы и упорядочивая поля в областях фильтра, рядов, категорий и данных.

Открыть в режиме сводной таблицы или сводной диаграммы можно таблицу, запрос или форму. Использование здесь термина «открыть» несколько необычно, так как при первом открытии таблицы или формы никакие данные в результатную таблицу/диаграмму не включаются. Правильнее было бы назвать это формулированием запроса. Возможность создания сводной таблицы (диаграммы) можно считать элементами OLAP-технологии.

На рис. 8.39 показана форма «Сотрудник», первый раз открытая в режиме сводной таблицы.

* Такая возможность появилась начиная с версии Access 2002.

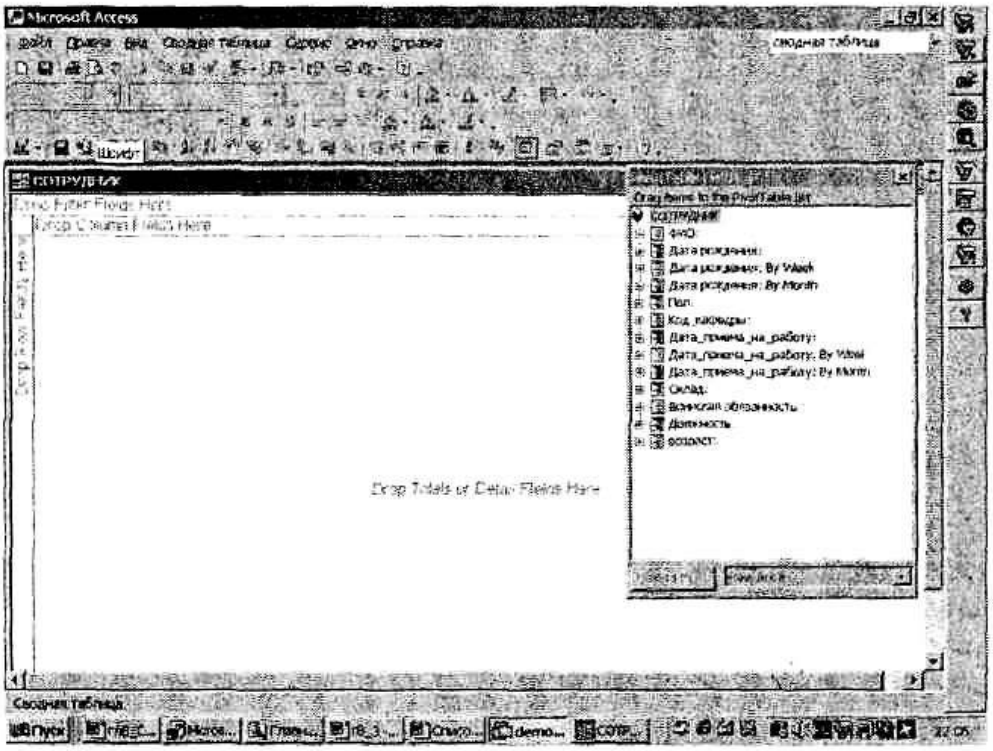


Рис. 8.39. Вид окна формирования сводной таблицы при первоначальном открытии

В режиме сводной таблицы имеется возможность добавлять поля в областях фильтра, строк, столбцов и данных. Можно также перемещать поля из одной области в другую и удалять поля из представления.

Предположим, что мы хотим проанализировать зарплату сотрудников в разрезе конкретных кафедр. В качестве источника для создания формы используем запрос, изображенный на рис. 8.40.

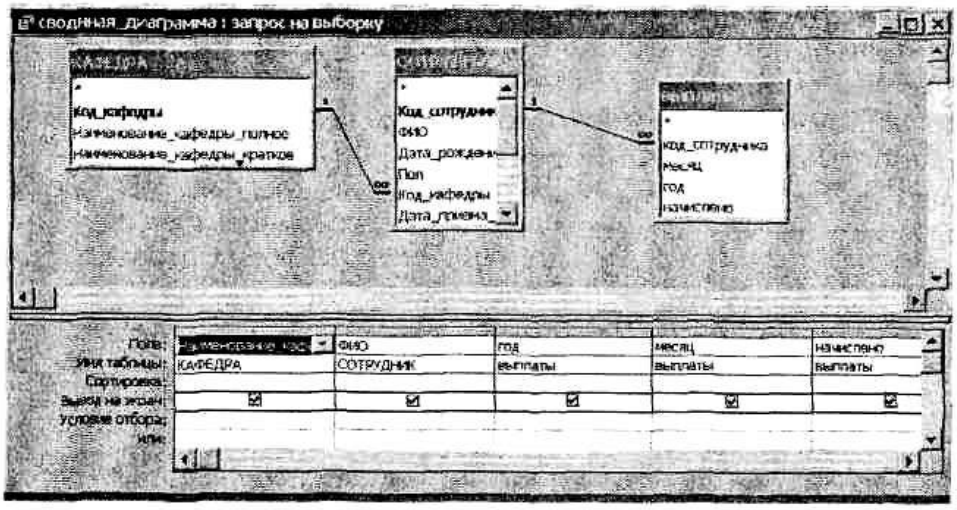


Рис. 8.40. Запрос, использующийся при формировании сводной таблицы

Вид сводной таблицы, построенной на основе этого запроса, представлен на рис. 8.41, а сводной диаграммы - на рис. 8.42.

Наименование конкретной кафедры, которое высвечивается вверху, легко может быть заменено путем выбора из списка имеющихся кафедр на другое. Так же может быть выбран конкретный период, за который мы хотим проанализировать данные.

Для графического представления информации был выбран тип обычной столбиковой диаграммы (см. рис. 8.42). Access предоставляет широкий выбор типов диаграмм, из которого можно подобрать подходящий для каждого конкретного случая.

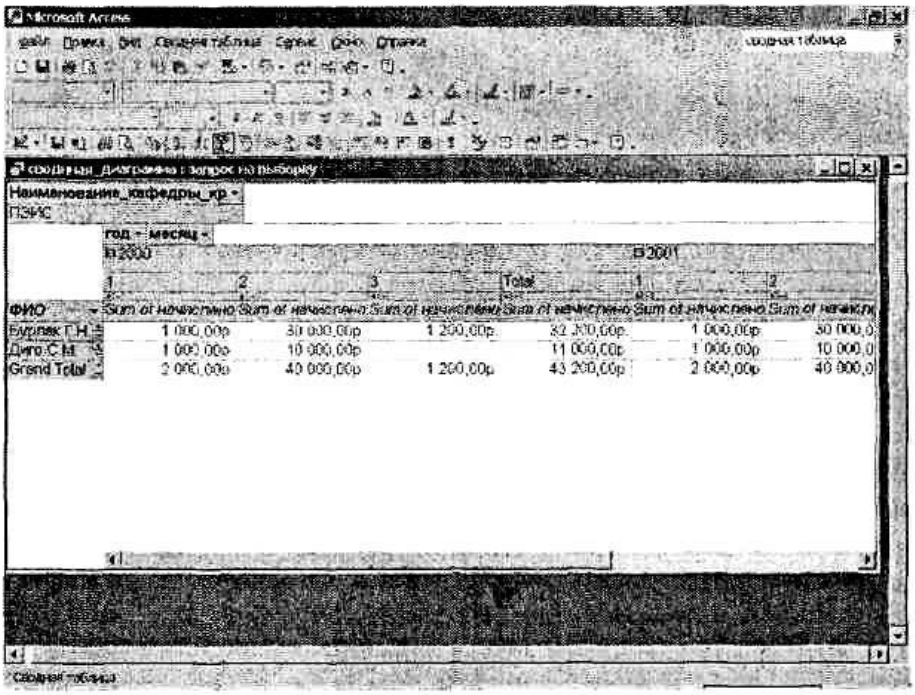


Рис. 8.41. Вид сводной таблицы

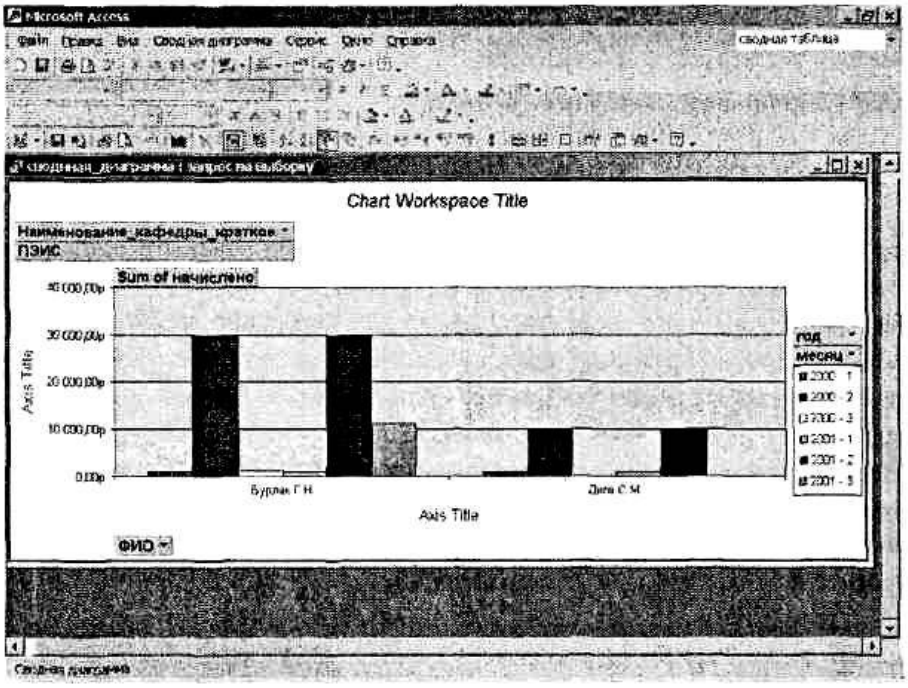


Рис. 8.42. Вид сводной диаграммы

8.3.7. Создание страниц доступа

В современных информационных системах важными являются возможности публикации данных в Интернете или Интранете. Начиная с Access 2000 такую возможность обеспечивают *Страницы доступа к данным*.

Если страница доступа к данным создается при открытой базе данных, то для этой страницы создается ярлык, который отображается в окне базы данных. В противном случае создается обособленная страница.

Для создания страницы следует в окне базы данных позиционироваться на позиции «Страницы» (см. рис. 5.4). Страница доступа к данным может быть создана с помощью Мастера или в режиме **Конструктор**. Создание страницы доступа с помощью Мастера похоже на создание формы или отчета: так же определяется источник данных (таблицы или запросы) и выбираются поля, которые добавляются на страницу. При необходимости определяются уровни группировки и порядок сортировки записей.

Страницы доступа к данным хранятся вне файла базы данных Access и могут публиковаться на Web-сервере.

Страницы доступа к данным отображают текущие данные из подключенной базы данных и являются интерактивными. Страницы могут распространяться в электронном виде с помощью электронной почты. Получатели будут видеть текущие данные при каждом открытии сообщения.

На это следует обратить внимание

- Генераторы экранных форм являются мощным средством автоматизации проектирования информационных систем.
- Экранные формы обеспечивают интерфейс пользователя с информационной системой и должны быть тщательно спроектированы, чтобы обеспечивать удобство, эффективность и безопасность работы.
- Современные генераторы экранных форм позволяют получать экранные формы разной степени сложности, использовать разные элементы управления, обладают развитыми возможностями по оформлению; задача проектировщика - осуществить правильный выбор возможностей.
- Экранные формы могут различаться по своему назначению. Подходы к проектированию экранных форм разного назначения имеют специфические особенности.
- Следует использовать единый стиль оформления экранных форм в рамках всей информационной системы.
- Генераторы экранных форм, включенные в состав конкретных СУБД, различаются по своим функциональным возможностям, удобству использования. Однако принципы, положенные в их основу, являются общими. Знание этих принципов позволяет легко осваивать разные реализации этих систем.

Контрольные вопросы

1. Какие разновидности экранных форм вы знаете?
2. Каково назначение экранных форм?
3. В каких режимах можно работать с экранной формой? Каково назначение каждого из этих режимов?
4. Какими способами можно создавать экранную форму?
5. Как можно включать поля таблицы/запроса в форму при создании формы с помощью Мастера?
6. Как можно включать поля таблицы/запроса в форму при работе в режиме Конструктор?
7. Как можно скорректировать ранее созданную экранную форму?
8. Какие элементы управления могут быть использованы в экранной форме? Какими способами можно создать эти элементы?
9. Каким образом можно менять размещение элементов на экране?
10. Как можно менять размер элемента управления?
11. Чем отличается элемент типа «Список» от «Поле со списком»? Как создается каждый из этих элементов? В каком случае какой тип элементов надо использовать?
12. Какие преимущества имеет использование элемента типа «Список» по сравнению с элементом типа «Поле со списком» при вводе данных в БД?
13. Как можно преобразовывать один тип элемента в другой? Какие преобразования допустимы, а какие - нет?
14. Какие разновидности многостраничных форм можно создавать в Access? Как их можно создавать?
15. Что такое «многотабличные» формы? Какие разновидности многотабличных форм можно создавать в Access? Как можно их создавать?
16. Что может являться источником данных для экранной формы?
17. Каковы особенности использования запроса со звездочкой «*» в качестве источника для формы?
18. Как можно включить в форму вычисляемое поле?
19. Как можно включить в форму рисунок?
20. Как можно запретить корректировку таблицы при ее просмотре с помощью формы?
21. Как можно создать форму для ввода данных?
22. К каким последствиям может привести корректировка таблиц, на которых базируется форма? Какие меры необходимо предпринять для устранения этих последствий?
23. Как можно включать в форму «кнопки»?
24. Как в Access можно создать «кнопочную» форму?
25. Что такое «страница доступа», как и с какой целью она создается?
26. Чем отличаются друг от друга подчиненные и связанные формы?
27. Как можно изменить последовательность обхода полей в форме? Для достижения каких целей это следует делать?
28. Как можно создать форму, используемую только для ввода данных?

29. Как можно создать форму только для просмотра?
30. Как можно создать форму, включающую несколько вкладок?

Глава 9 СОЗДАНИЕ ОТЧЕТОВ

9.1. Общая характеристика отчетов

Термин «отчет» понимается в ИС шире, чем это традиционно принято. Под *отчетом* здесь понимается любой выходной документ: список (например, сотрудников), письмо, адрес, печатающийся на конверте (почтовая этикетка), отчет в традиционном понимании этого слова. Создание отчетов (выходных документов) является одной из наиболее важных функций информационных систем.

Для создания отчетов используются высокоуровневые средства автоматизации - генераторы отчетов. Генераторы отчетов, так же как и генераторы форм ввода-вывода, являются компонентами языков 4-го поколения. Они включены в состав большинства СУБД. Кроме того, генераторы отчетов представлены и как самостоятельный класс программного обеспечения. Существует даже англоязычный термин «reporting», объединяющий все вопросы, относящиеся к процессу получения отчетов.

Чаще всего, когда отчеты обсуждаются в контексте БД, речь идет об извлечении информации из каких-либо источников (*однотипных* или *разнотипных*) и представлении их в виде, удобном для дальнейшего восприятия и анализа. Это делает рассматриваемые вопросы близкими к проблематике OLAP-систем.

Источниками для получения отчетов могут быть не только таблицы баз данных, но и запросы, а также записи, отобранные с помощью фильтров. Некоторые генераторы отчетов позволяют проводить отбор данных, включаемых в отчет, непосредственно пользуясь средствами самого генератора отчетов.

Отчеты позволяют выполнять следующие действия:

- проводить группировку данных;
- вычислять многоуровневые промежуточные и общие итоги по отдельным полям;
- вводить в отчеты вычисляемые поля;
- выводить в отчеты данные из разных источников;
- включать в отчеты данные, отобранные по заданным критериям;
- использовать различные формы представления информации;
- качественно оформлять выводимые данные.

Отчеты имеют много общего с формами. Однако отчеты в отличие от форм не предназначены для ввода и правки данных в таблицах. Они позволяют только выводить данные в различном виде. Вывод отчета может быть осуществлен на экран, на печать, а также в файл. Чаще всего отчеты используются для вывода информации на печать (т.е. для получения так называемых твердых копий). Для документов, даже одинаковых по содержанию, могут использоваться разные приемы их оформления в зависимости от того, куда осуществляется вывод информации. Например, при выдаче информации

на экран могут использоваться специальные эффекты (мигание, динамические изображения, полосы прокрутки и т. п.).

Отчеты могут быть разной формы (рис. 9.1)

Различают отчеты *анкетной* и *табличной* формы. При анкетной форме данные об одном объекте (сотруднике, товаре и т.п.) обычно размещаются один под другим, причем слева указывается название атрибута (поля), а справа - его значение. После вывода информации об одном объекте выводится информация о следующем объекте (рис. 9.2). Иногда такой тип документа называют документом в виде формы (действительно, это очень напоминает позаписный вывод информации на экран при использовании экранных форм).

Документы табличной формы включают в себя привычные таблицы с названиями атрибутов в заголовках столбцов; данные о каждом объекте представляются в одной строке (рис. 9.3).

Табличные документы могут включать либо одну таблицу - *однотабличные* документы, либо несколько таблиц (обычно разной структуры) - *многотабличные* документы.

В зависимости от объема и выбранного способа оформления документы могут занимать одну страницу (*одностраничные*) или несколько страниц (*многостраничные*).

В любом документе могут быть выделены самостоятельные разделы (зоны, «полосы», области), выполняющие разную смысловую нагрузку. В реальном документе те или иные из перечисленных ниже зон могут отсутствовать. Различают зоны, относящиеся целиком к документу; это *заголовок отчета* и *примечание документа* («итоговая» зона, оформительская часть). Заголовок отчета и примечание отчета могут размещаться на первой и последней страницах отчета соответственно либо выноситься на отдельные листы.

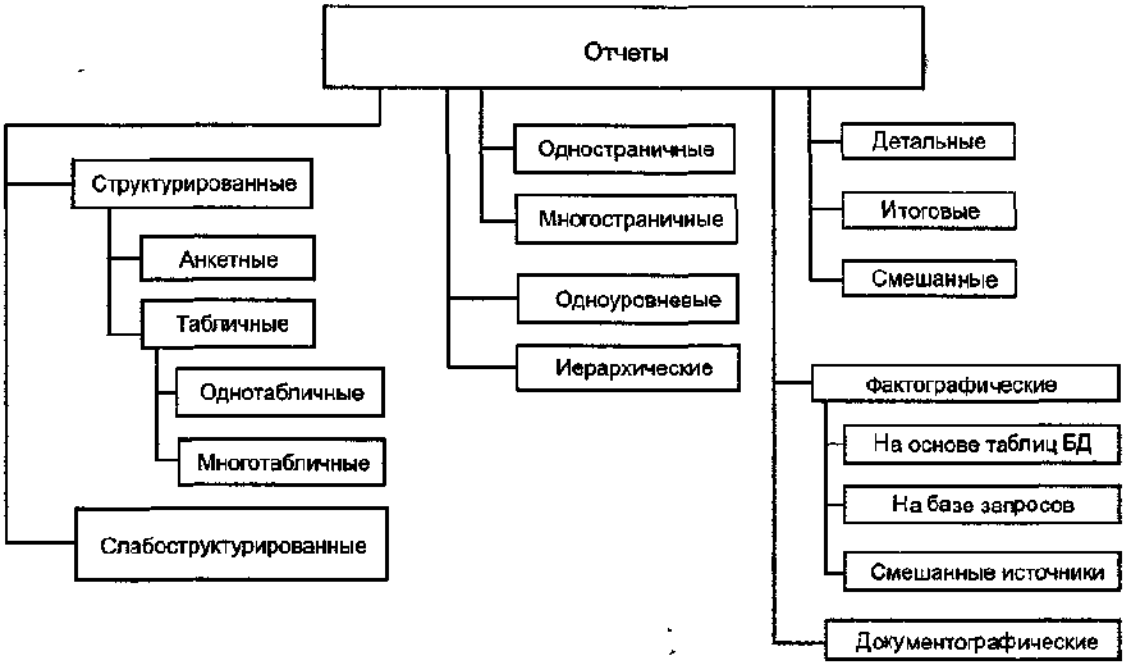


Рис. 9.1. Классификация отчетов

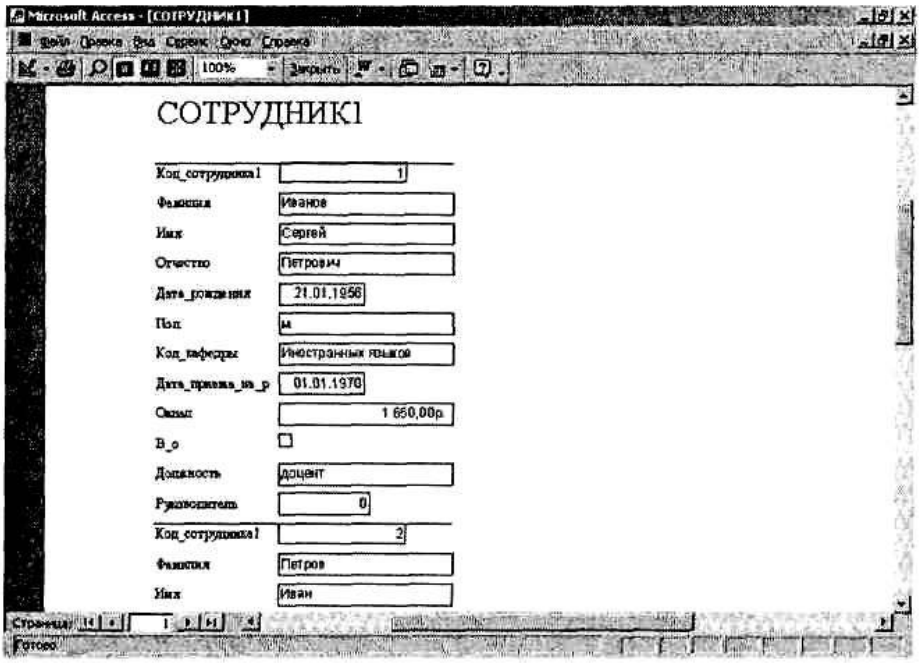


Рис. 9.2. Фрагмент документа анкетной формы

Ведомость на выплату зарплаты
Отдел_____
Месяц_____

№ п/п	ФИО	Сумма на руки	Подпись получателя

Рис. 9.3. Пример документа табличной формы

Для каждой страницы выделяют *верхний* и *нижний колонтитулы* (заголовок и «подножие» страницы). Ну и, естественно, главное место принадлежит *области данных*. В этой области размещаются данные из БД.

Кроме того, в документе может быть обеспечена группировка данных, причем в отличие от запросов группировка может быть одноуровневой и многоуровневой (иерархической). В последнем случае для каждого уровня группировки могут быть созданы *зоны заголовка* и *примечания группы*.

Группировка обычно используется в целях подсчета каких-либо итоговых показателей для каждой группы (суммы, количества элементов в группе и т.п.). При этом возможно получение *итоговых* документов, включающих только итоговые значения, *детальных* документов, имеющих только детальные строки, и *смешанных*, содержащих как детальные строки, так и итоговые.

Отчеты с подведением итогов могут быть использованы и при создании документов анкетной формы. Например, при выдаче в анкетной форме сведений о сотрудниках в конце документа можно подсчитать общее количество сотрудников. Но поскольку наиболее часто эти возможности используются в табличных документах, то соответствующие классификационные группировки отнесены именно к этому виду документов.

Источниками информации для отчетов могут быть либо реальные таблицы базы данных, либо предварительно созданные запросы, отбирающие информацию, выводимую в отчет. Кроме того, в отчет могут включаться вычисляемые поля. Вычисляемые поля, как, впрочем, и реальные поля БД, могут входить в любую зону документа.

В последнее время в отчеты, наряду с *символьной* информацией, часто включается *деловая графика*.

Кроме документов, содержащих главным образом *фактографическую* информацию из баз данных, можно создавать и документы, которые в основном, напротив, включают какой-то текст (*документографические*), в который вкраплены данные из БД (*документы типа письма*).

В документах фактографического типа можно различать просто какой-то текст, не имеющий жесткой связи с элементами БД (например, название документа, поясняющий текст), названия элементов из БД (например, «Фамилия») и значения этих элементов (например, Иванов, Петров), элементы оформления (линия, рисунки).

Генераторы отчетов разных СУБД различаются по своим возможностям и особенностям выполнения идентичных функций. Так, например, в некоторых системах в итоговую зону документа по умолчанию включаются суммарные величины всех числовых полей, включенных в отчет. Необходимо внимательно проанализировать отчет, полученный в результате применения Мастера, и внести необходимые изменения в форму отчета.

Перечни типов отчетов, которые предлагает та или иная СУБД, могут различаться. Так, например, в dBase есть тип отчета «Письмо» (Mail) [18]. Суть его заключается в том, что при выборе этого типа отчета открывается область, в которую можно вводить текст «письма». В текст этого письма могут включаться значения тех или иных полей из базы данных. В шаблоне письма при этом будут указываться ссылки на соответствующие поля, а при выводе этих документов на экран или на печать эти ссылки заменяются соответствующими значениями полей. Подобное использование баз данных наблюдается довольно часто, и не только при создании «писем» в узком смысле этого слова. Такой тип отчетов имеется далеко не во всех СУБД. Например, в Access такой тип отчетов отсутствует, но это не означает, что нельзя получить документ такого типа. Просто нужно воспользоваться другими возможностями, в частности возможностью создания составного документа путем слияния.

9.2. Создание отчетов в системе Access

Для создания отчета в окне базы данных следует перейти на вкладку Отчеты, щелкнуть по кнопке Создать и в верхней части появившегося окна Новый отчет (рис. 9.4) выбрать способ создания отчета, а в нижней - указать таблицу или запрос, данные из которого будут выводиться в отчете. После этого следует щелкнуть по кнопке ОК.

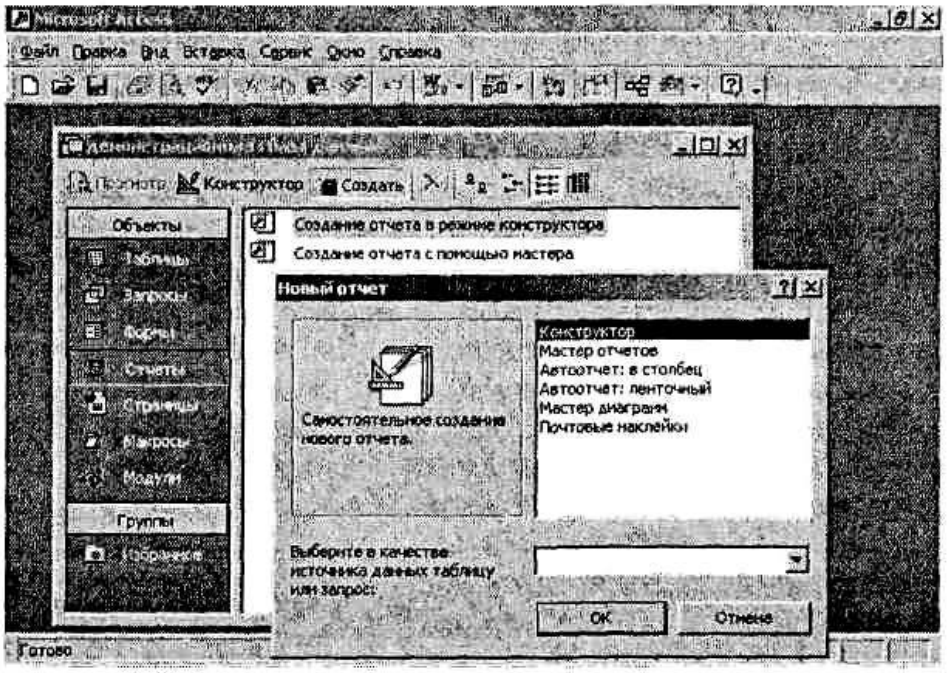


Рис. 9.4. Начальное окно создания нового отчета

9.2.1. Выбор способа создания отчета

В окне Новый отчет можно выбрать любую из предлагаемых системой возможностей. Но если при создании таблиц и запросов основным средством является Конструктор, то с отчетами, как и с формами, дело обстоит иначе. Гораздо удобнее воспользоваться Мастером или создать какой-нибудь автоотчет, а затем с помощью Конструктора внести в него необходимые изменения. В принципе в данном случае Мастерами можно считать все возможности, перечисленные в ниспадающем меню, изображенном на рис. 9.4, кроме позиции **Конструктор**.

Простейшим способом создания отчетов является использование «Автоотчетов». При использовании этой возможности в отчет выводятся все поля выбранного источника данных (таблицы/запроса), названием отчета и его заголовком становится название источника данных. Никакое вмешательство пользователя в процесс создания документа не предусматривается.

Более гибкой возможностью является применение **Мастера отчетов** (рис. 9.5). Именно этот инструмент наиболее часто используется при первоначальном создании документа. При любом способе создания отчета необходимо выбрать источник данных, на основе которого он будет формироваться. Источником может быть одна (или несколько) таблица либо запрос.

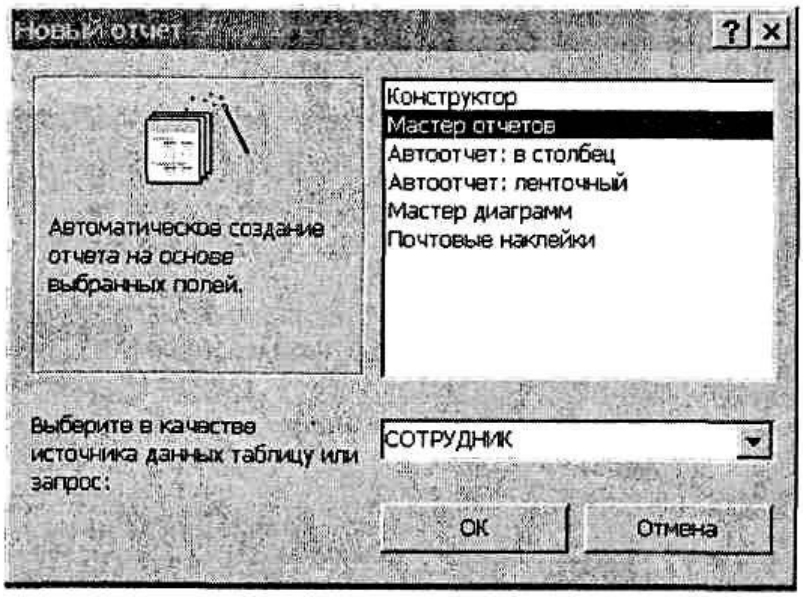


Рис. 9.5. Проектирование отчетов. Создание нового отчета. Выбор способа создания и источника данных

9.2.2. Создание отчетов с использованием Мастера отчетов

Рассмотрим возможность создания отчетов с использованием Мастера отчетов. Начальный вид экрана **Создание отчетов** в режиме Мастера представлен на рис. 9.6. Если вы забыли выбрать источник отчета (таблицу или запрос) на первом шаге или если в качестве источника требуется выбрать несколько таблиц/запросов, то это можно сделать и на данном этапе, выбрав нужные источники в окне **Таблицы и запросы**. Далее необходимо выбрать поля, которые войдут в отчет. При выборе этих полей можно либо использовать кнопку с двумя стрелками (в случае, если в отчет будут входить все поля или их большинство), либо переносить поля по одному, используя кнопку с одной стрелкой. При этом следует обратить внимание на то, что поля нужно переносить не в том порядке, в котором они располагаются в структуре исходной таблицы, а в том, в котором они будут использоваться в отчете. И хотя потом, в режиме **Конструктор**, можно менять положение тех или иных элементов в отчете, лучше сразу продумывать эти вопросы.

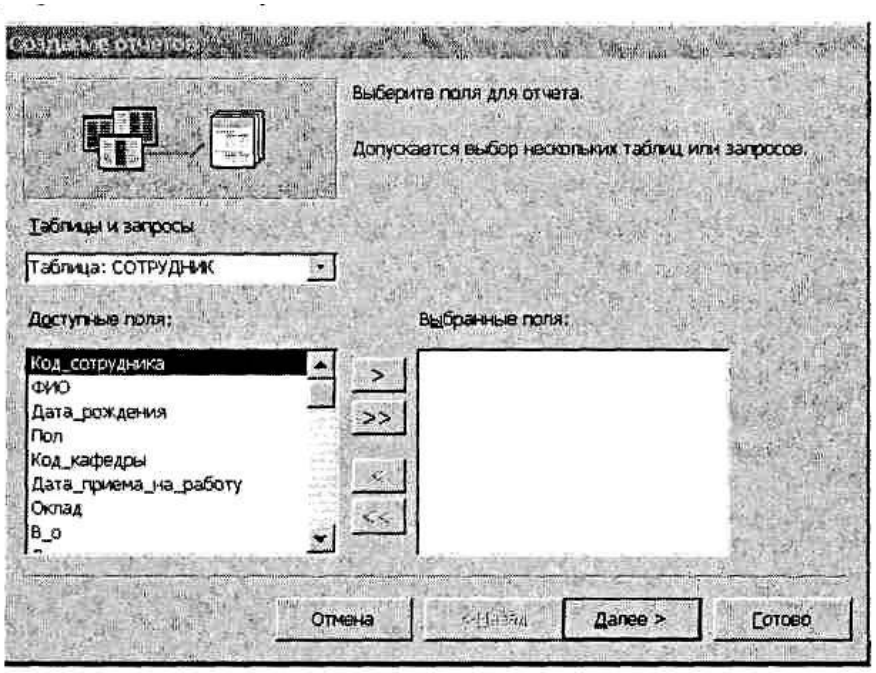


Рис. 9.6. Начальный вид окна **Создание отчетов** при использовании Мастера

Предположим, что требуется на основе таблицы «Сотрудники» получить ведомость на выдачу зарплаты. Для простоты считаем, что все сотрудники получают фиксированный оклад. Информация сгруппирована по кафедрам. Внутри кафедры записи упорядочены по алфавиту по полям «Фамилия», «Имя», «Отчество».

После выбора полей, входящих в наш отчет, окно **Создание отчетов** приобрело вид, изображенный на рис. 9.7.

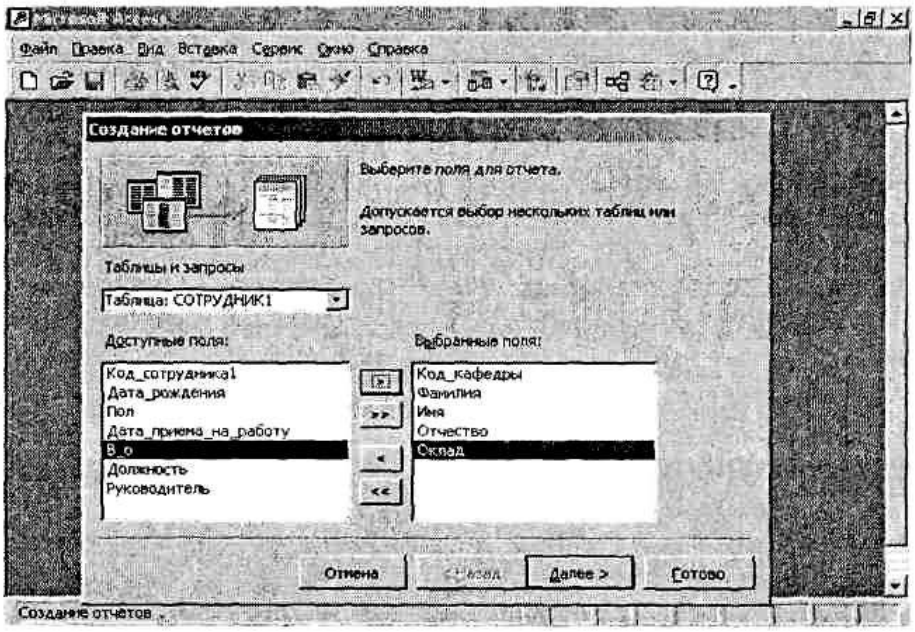


Рис. 9.7. Вид окна **Создание отчетов** при использовании Мастера после выбора полей, входящих в отчет

После определения полей, включаемых в отчет, система предлагает выбрать уровни группировки (рис. 9.8). В нашем примере нас устраивает предложенное поле группировки, и новых уровней группировки вводить не надо, поэтому щелкнем по кнопке. Окно примет вид, изображенный на рис. 9.9.

В общем случае для выделения уровней группировки нужно позиционироваться на соответствующем поле (поле группировки) и щелкнуть по кнопке со стрелкой. Если предполагается несколько уровней группировки, то поля должны выбираться в порядке старшинства группировки.

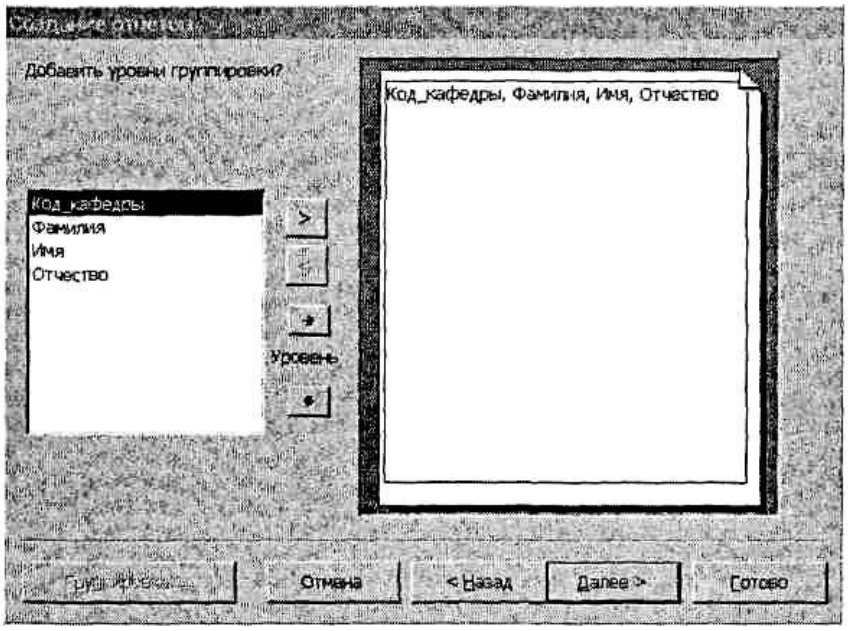


Рис. 9.8. Вид окна **Создание отчетов.**
Определение уровней группировки (экран 1)

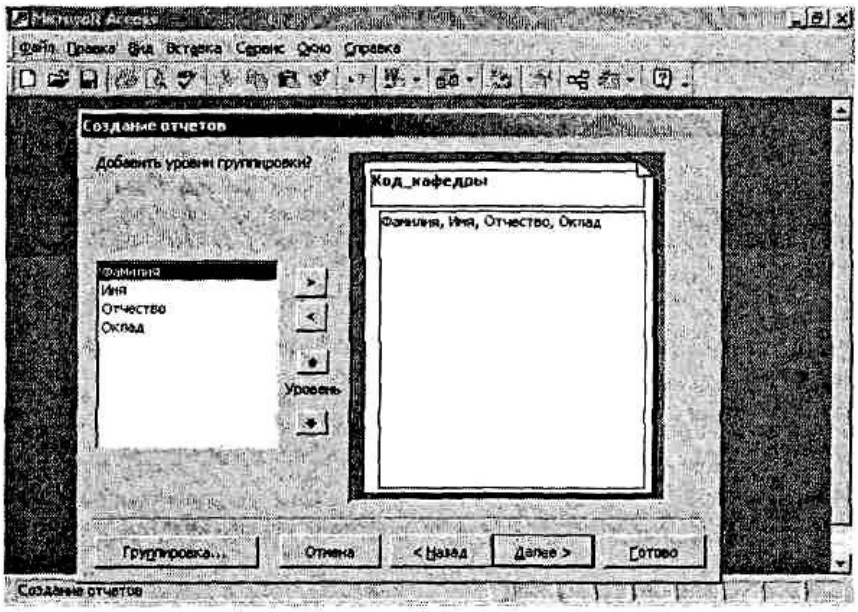


Рис. 9.9. Вид окна **Создание отчетов.**
Определение уровней группировки (экран 2)

Следующий, третий шаг (рис. 9.10) позволяет задать порядок сортировки. Мы выбрали сортировку по трем полям («Фамилия», «Имя», «Отчество»).

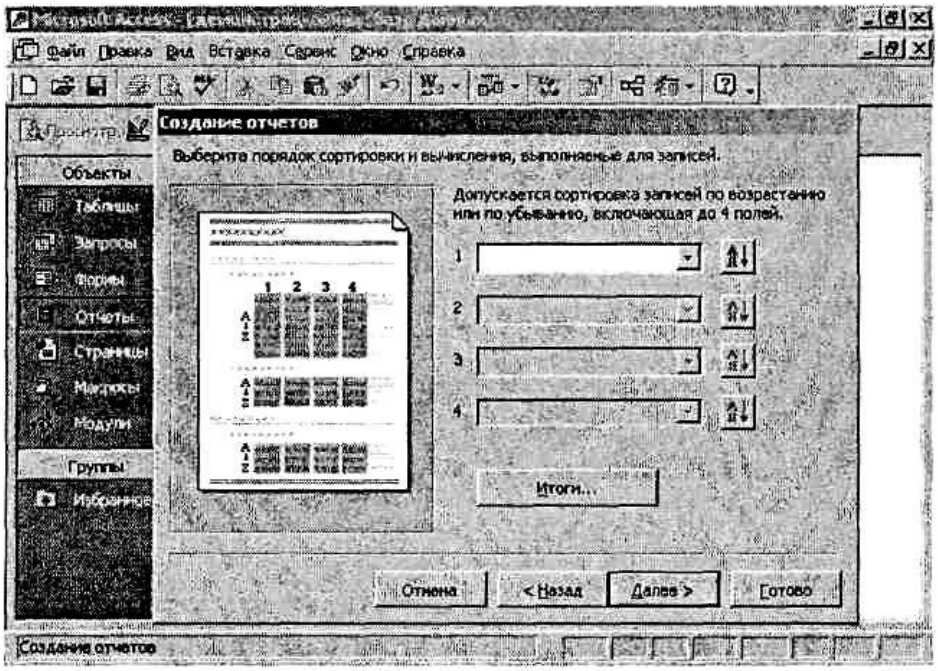


Рис. 9.10. Вид окна **Создание отчетов.**
Определение порядка сортировки

Если необходимо вычисление итогов, то следует щелкнуть по кнопке **Итоги**, после чего высветится экран (рис. 9.11), на котором будут перечислены все поля числового типа, включенные в отчет (в нашем примере такое поле только одно - «Оклад»). Необходимо выбрать те поля, по которым будут подводиться итоги, а также вид агрегирующей функции. Мы выбрали получение суммарных итогов (Sum) по полю «Оклад».

Далее надо выбрать вид макета отчета (рис. 9.12), задать стиль отчета (рис. 9.13), а также имя отчета (рис. 9.14). Если имя отчета не задано, то по умолчанию будет задано имя, совпадающее с именем таблицы, на основе которой формируется отчет.

Следует обратить внимание, что заданное имя отчета выступает в двух ролях: как имя объекта-отчета и как название документа, выводимое в заголовке отчета. В нашем примере документ должен назваться «Ведомость на получение зарплаты». Такие длинные имена объектам обычно не присваиваются. Если переименовать объект **Отчет** после его создания, то заголовок документа не изменится. Переименовать его можно обычными способами, например, выделив название отчета в окне базы данных, после чего нажать правую клавишу мыши, в появившемся меню выбрать позицию «***переименовать***» и набрать новое имя. Как изменить заголовок отчета, будет рассмотрено ниже.

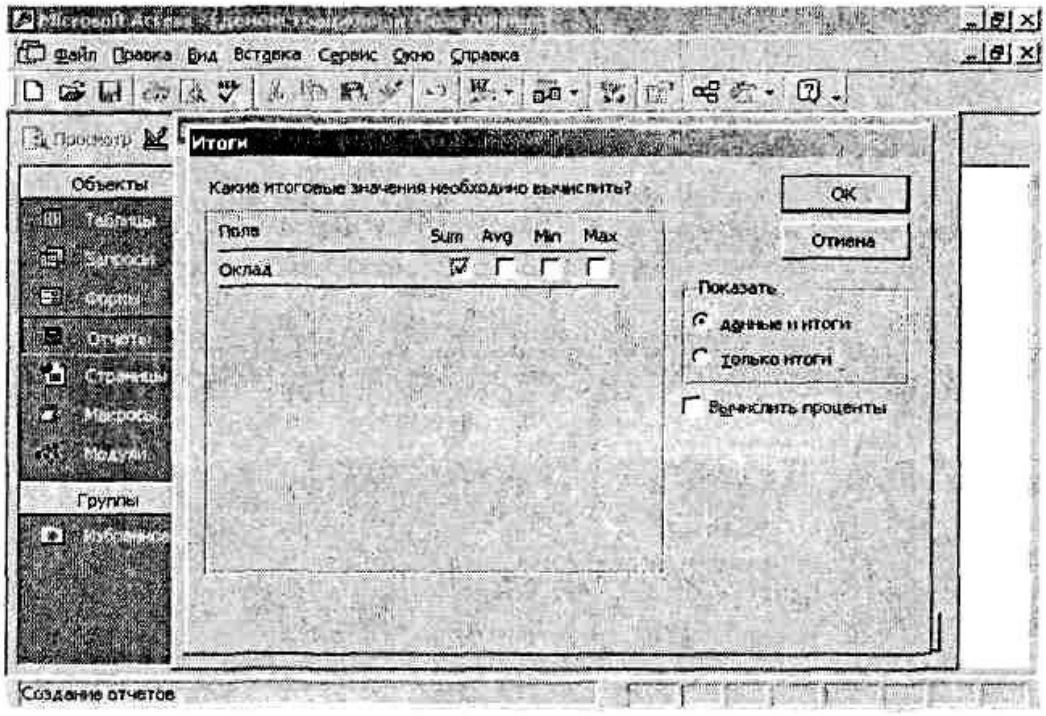


Рис. 9.11. Возможность задания итогов

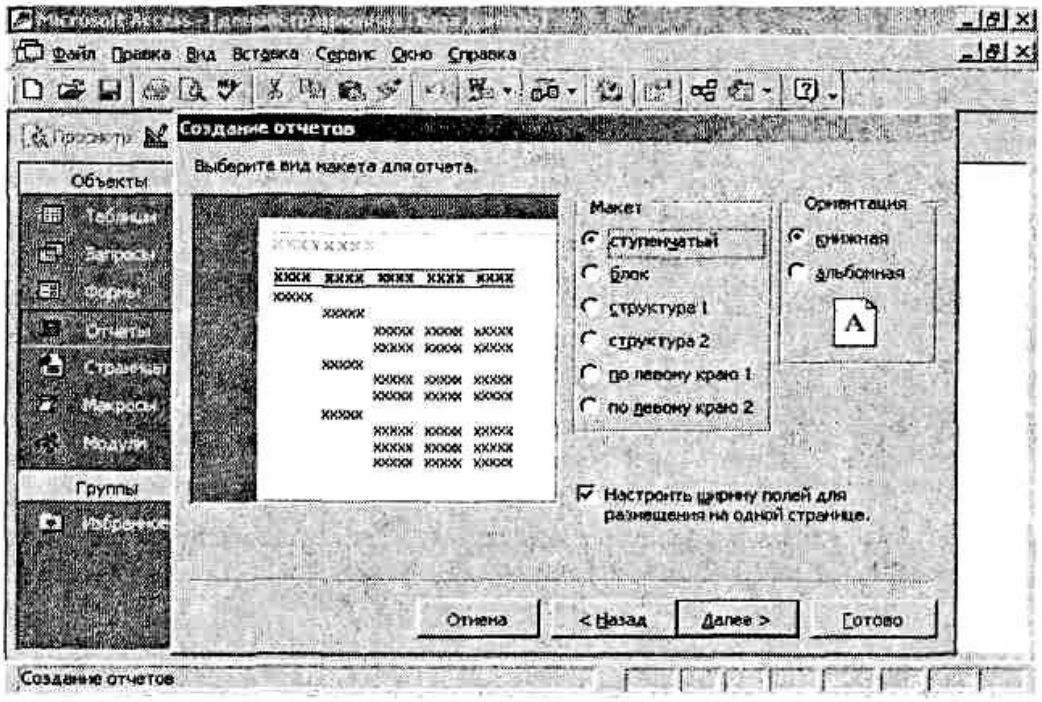


Рис. 9.12. Вид макета отчета

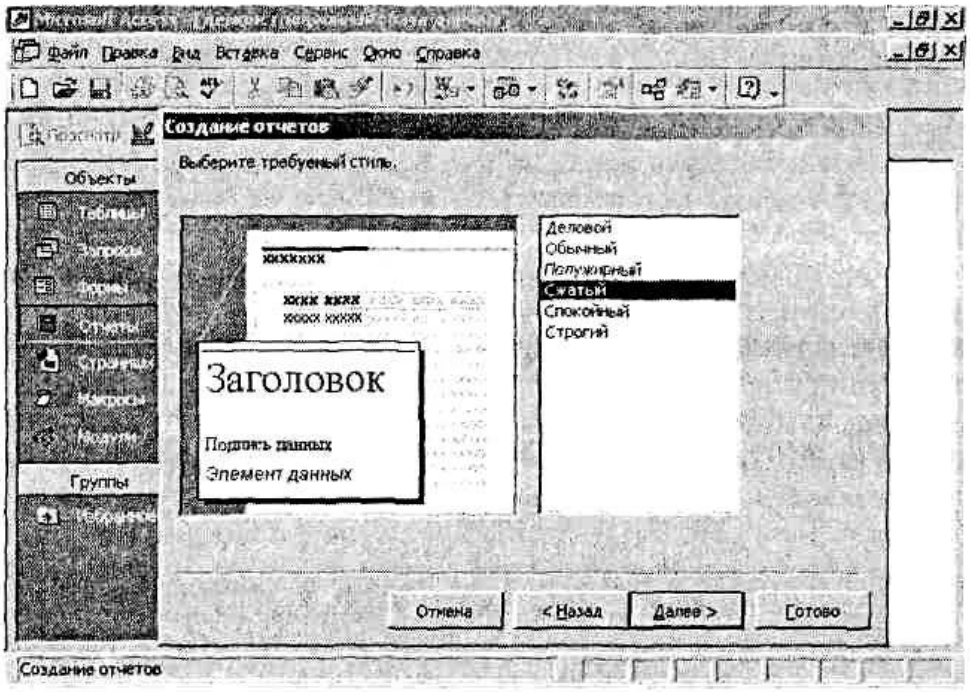


Рис. 9.13. Задание стиля отчета

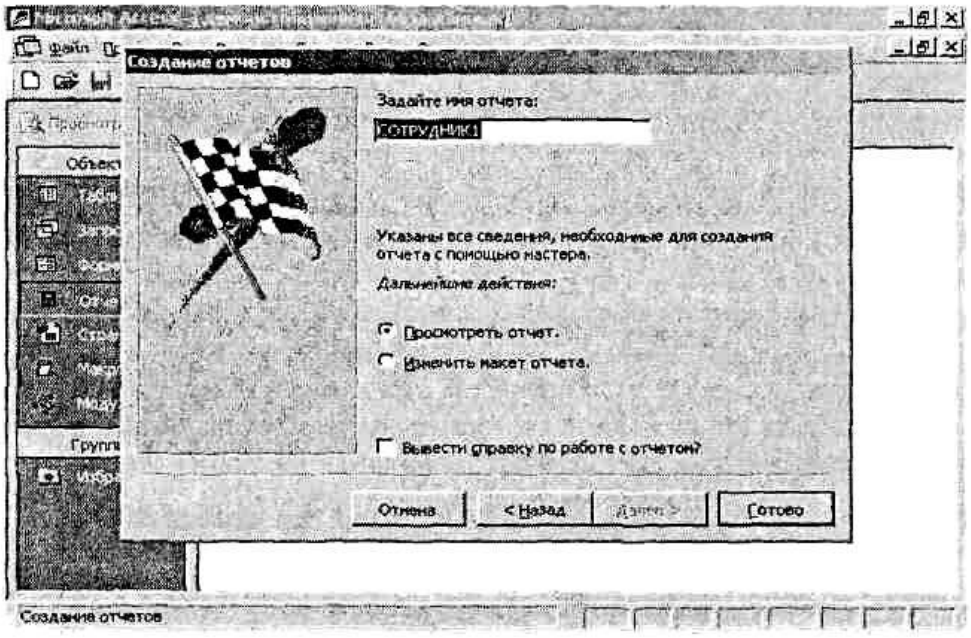


Рис. 9.14. Задание имени отчета

Кроме того, хотелось бы обратить внимание, что здесь мы рассматриваем некоторые возможности генераторов отчетов как инструментального средства и не касаемся других вопросов (проектирование структуры базы данных, проектирование форм документов, алгоритмы и пр.). Чтобы примеры были обозримыми, реальные ситуации сильно упрощены. Так, чтобы реально определить начисленную зарплату, даже если все сотрудники работают на окладе, необходимо определять, полный ли месяц отработал сотрудник (т.е. когда он принят на работу, пропускал ли работу по болезни или по каким-либо другим причинам, не был ли он в отпуске), не изменялся ли у него

в данный период оклад и др. Естественно, что и сами таблицы должны быть спроектированы несколько иначе, чем в рассматриваемом примере.

9.2.3. Корректировка отчета в режиме Конструктор

Переход в режим Конструктор

Полученный в результате проведенных выше действий отчет имеет вид, представленный на рис. 9.15. Естественно, что он нуждается в корректировке. В нашем случае прежде всего следует изменить название отчета*, а также подписи для итоговых показателей.

Для того чтобы изменить полученный документ, необходимо перейти в режим конструктора отчетов. Для этого в пункте меню **Вид** можно выбрать команду **Конструктор** либо соответствующую кнопку на панели инструментов.

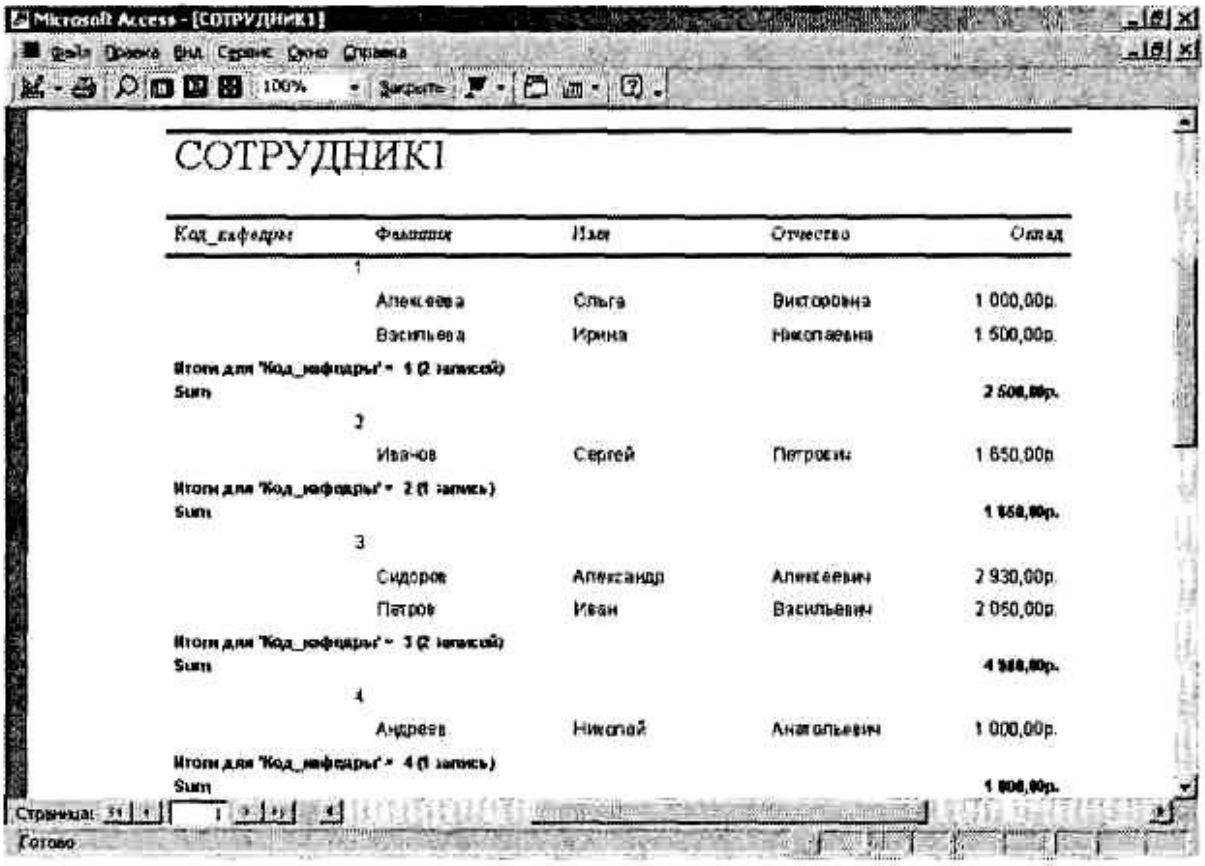


Рис. 9.15. Вид полученного отчета

В режиме Конструктор полученный документ имеет вид, показанный на рис. 9.16.

* На последнем шаге создания отчета можно было задать и другое имя отчета, в том числе и совпадающее с названием документа, которое указывается в его заголовочной части; в этом случае корректировки заголовка отчета не потребовалось бы.

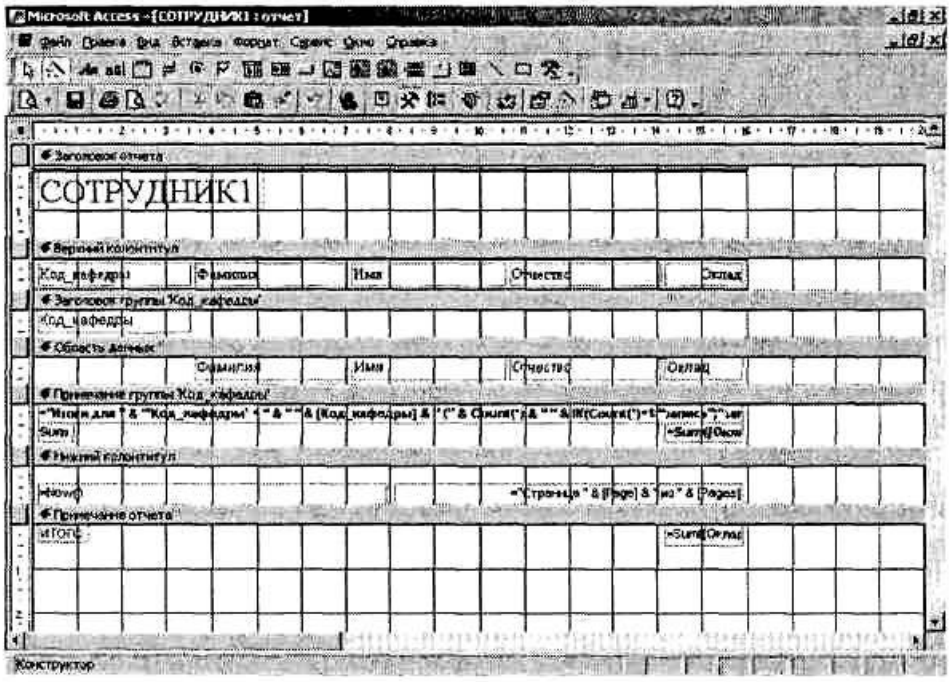


Рис. 9.16. Вид документа в режиме **Конструктор**

При работе в режиме **Конструктор** на экране появляются три панели, используемые при создании/корректировке отчетов: «Конструктор отчетов», «Формат (форма/отчет)» и «Панель инструментов». Каждая из этих панелей может быть отключена. На рис. 9.16 отсутствует панель «Формат».

Часть кнопок «Конструктора отчетов» является общей для многих офисных Windows-приложений и знакома большинству пользователей. Поэтому рассмотрим назначение только тех кнопок, которые являются специфическими, предназначенными именно для построения отчетов:

- Сортировка и группировка - выводит на экран соответствующее окно, в котором указываются поля, используемые для группировки и сортировки данных в отчете;
- Панель элементов - включает и выключает режим показа в конструкторе отчетов панели инструментов;
- Автоформат - выводит на экран диалоговое окно, позволяющее изменить внешний вид всего отчета в целом;
- Свойства — выводит на экран окно свойств выделенного в данный момент в Конструкторе элемента. Каждое поле отчета обладает большим числом свойств. Некоторые из них будут рассмотрены ниже;
- Построить - используется для вызова нужного построителя (выражений, макросов или программ).

Панель инструментов, которая отображена на рис. 9.16, полностью совпадает с одноименной панелью, используемой при создании экранных форм. Как отмечалось выше, отчеты могут выводиться не только на печать, но и на экран. В связи с этим надо обратить внимание на то, что некоторые кнопки, включенные в данную панель, имеет смысл использовать только при создании экранных отчетов (это прежде всего относится к раскрывающимся спи-

скам, управляющим кнопкам и т.п.). Эти кнопки уже рассматривались при описании экранных форм, и в данном разделе мы к ним обращаться не будем.

Наиболее часто при создании/корректировке отчетов используются кнопки:

- элементы надписи - попадают в отчет в том виде, в каком они представлены в «Конструкторе отчетов»;
- элементы поля - указываются имена тех полей таблицы или запроса, данные из которых выводятся в отчете.

Корректировка отчета

Позиционируясь на соответствующем месте отчета и воспользовавшись кнопкой **Надписи**, изменим название отчета и сделаем его «Ведомость на выдачу зарплаты».

Далее изменим выражение

="Итоги для " & ""Код__ кафедры" = " & "" & [Код_ кафедры] & " (" & Count(*) & " " & 11f(Count(*)=1;"запись";"записей") & ")",

Sum

=Sum([Оклад

])

записанное в зоне «примечание для группы», на

="Итоги для кафедры"&" " & [Код_ кафедры]

=Sum([Оклад

]).

Тем самым мы откорректировали подписи и отказались от подсчета числа записей в группе. Отчет в режиме **Конструктор** теперь имеет вид, представленный на рис. 9.17, а в режиме предварительного просмотра - на рис. 9.18.

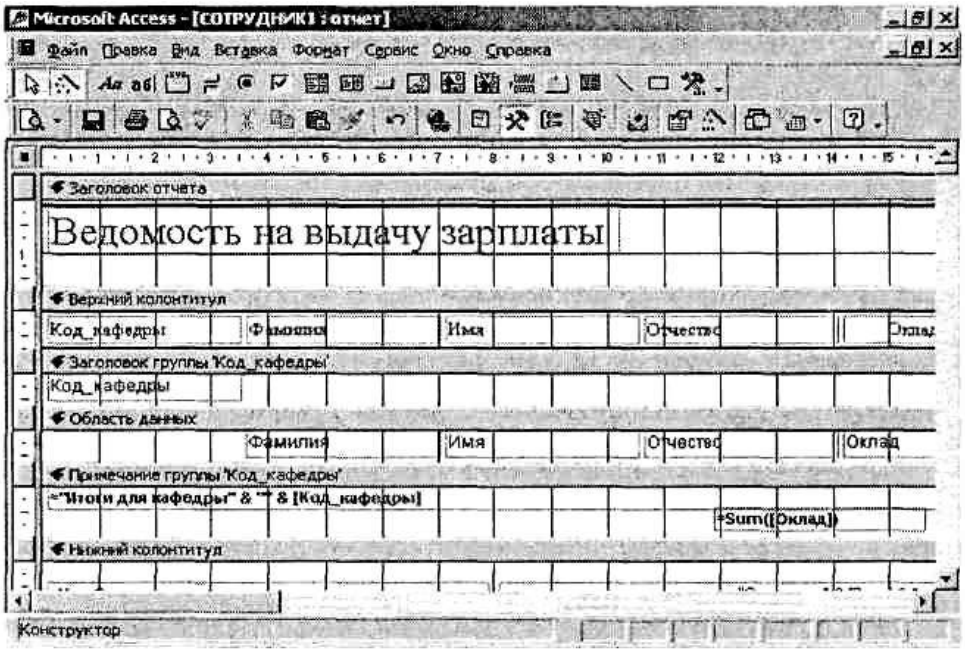


Рис. 9.17. Вид скорректированного отчета в режиме **Конструктор** (шаг 1)

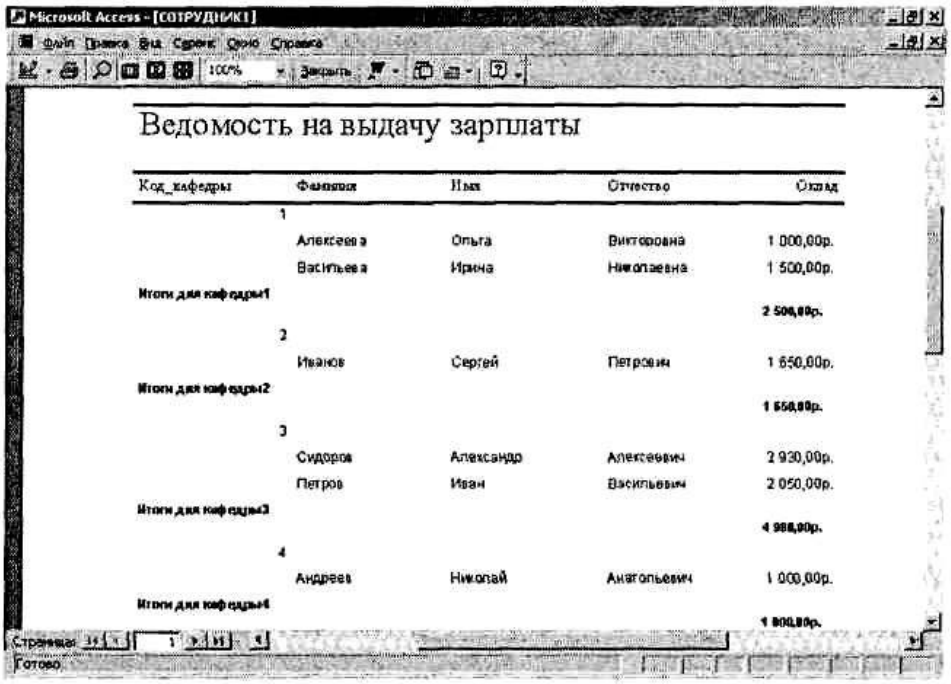


Рис. 9.18. Вид скорректированного отчета в режиме предварительного просмотра (шаг 1)

Еще раз обратимся к отчету в режиме конструктора. Поскольку мы при формировании отчета запросили получение одной степени итогов, то в нашем отчете имеем следующие разделы (области):

- Заголовок отчета;
- Верхний колонтитул;
- Заголовок группы «Код кафедры»;

- Область данных;
- Примечание группы «Код кафедры»;
- Нижний колонтитул;
- Примечание отчета.

Высоту области можно изменять, используя маркеры, расположенные на вертикальной линейке. Для этого подводят указатель мыши к маркеру и, когда он принимает вид двусторонней стрелки, при нажатой левой кнопке мыши перетаскивают его в нужном направлении.

Если в окне находится меньшее количество областей, чем вам нужно, то в пункте меню **Вид** следует отметить строки, соответствующие нужным областям отчета. Аналогичным образом могут быть удалены из отчета ненужные области.

Данные, находящиеся в области заголовка отчета, выводятся в начале первой страницы отчета. Как правило, в этой области помещают название отчета.

Данные, находящиеся в области верхнего колонтитула, выводятся в начале каждой страницы многостраничного отчета. На первой странице данные из области верхнего колонтитула выводятся вслед за данными из области заголовка отчета. В этой области при создании отчетов табличной формы обычно размещают названия столбцов, которые определяют содержание выводимых в отчете данных.

В области данных в режиме конструктора отображаются названия полей базовой таблицы (запроса). В самом отчете вместо них будут выводиться данные, хранящиеся в этих полях. Данные, размещенные в этой области, выводятся в отчете строка за строкой. В области данных, как, впрочем, и в любой другой области, также могут быть размещены вычисляемые поля.

Элементы, размещенные в области нижнего колонтитула, выводятся внизу каждой страницы отчета. Часто это бывают номера страниц либо какие-то элементы оформления.

В рассматриваемом нами примере в конце каждой страницы помещается текущая дата (см. зону «Нижний колонтитул» на рис. 9.16), а также номер страницы и общее число страниц в отчете. На рис. 9.17 эти данные просто не видны, поскольку на них изображена не вся страница, а только ее фрагмент.

В примечании отчета помещают, как правило, итоговые значения (в нашем примере - суммарный размер выплаченной зарплаты), подписи и другую информацию, которую вы хотите поместить в конце отчета. Данные из этой области выводятся на последней странице отчета.

Для облегчения создания отчета справа и сверху выводятся линейки, позволяющие установить размер соответствующего элемента. Размер области можно изменять, если подвести указатель мыши к границе области, и, когда он примет вид двусторонней стрелки, при нажатой левой кнопке мыши перетаскивать границу области в нужном направлении.

Продолжим корректировку полученного отчета, демонстрируя тем самым другие возможности генератора отчетов. Прежде всего перенесем на-

звание поля «Код_кафедры» в зону «Заголовок_группы». Для этого необходимо активизировать данный элемент отчета, подведя указатель мыши к любому месту выбранного элемента, и один раз щелкнуть левой кнопкой мыши (вокруг активных элементов появляются маркеры - маленькие черные квадратики по углам и в центре каждой из сторон), после чего добиться, чтобы указатель мыши принял форму «ладошки», нажать левую клавишу мыши и, не отпуская ее, перенести элемент на нужное место (предварительно передвинув вправо элемент-поле «Код_кафедры» в этой зоне чуть правее, чтобы освободить место для расположения надписи). После этого изменим размер обоих элементов в зоне заголовка группы, чтобы они отображались более компактно. Изменить размер элемента можно следующим образом: выделить элемент; позиционировать указатель мыши так, чтобы он принял форму двунаправленной стрелки, нажать левую клавишу мыши и перетащить стрелку в нужном направлении до достижения требуемого размера элемента. Наш отчет в режиме конструктора теперь имеет вид, представленный на рис. 9.19, а в режиме предварительного просмотра - на рис. 9.20.

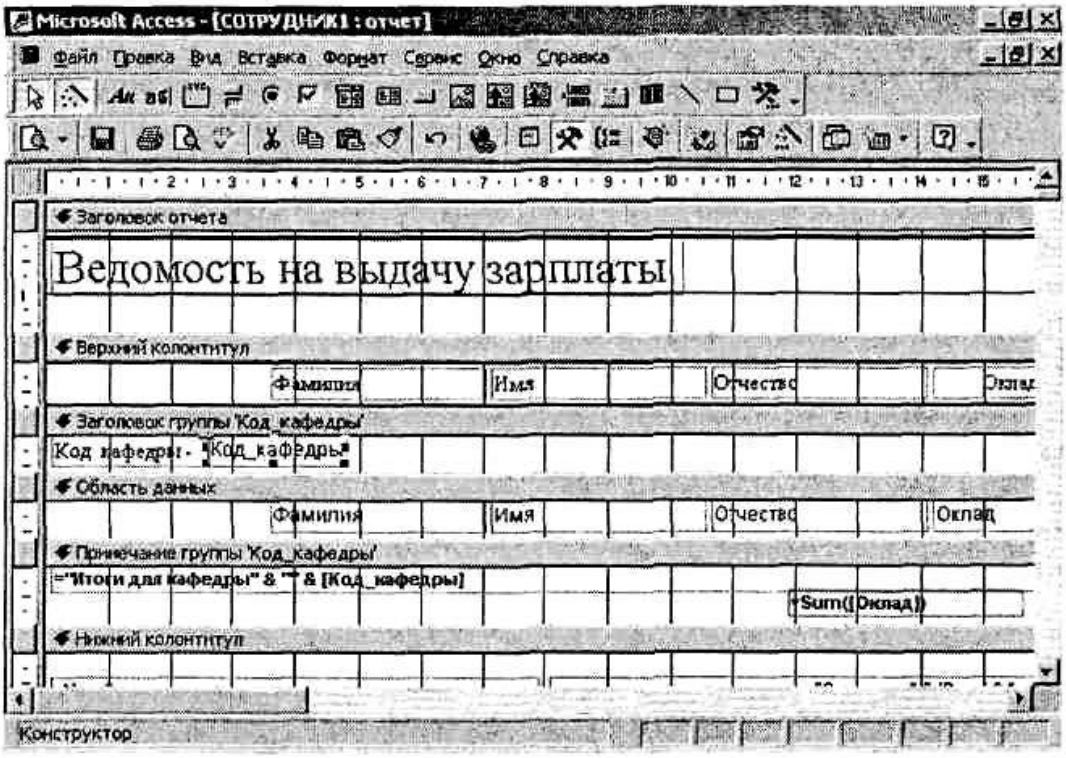


Рис. 9.19. Вид скорректированного отчета в режиме **Конструктор** (шаг 2)

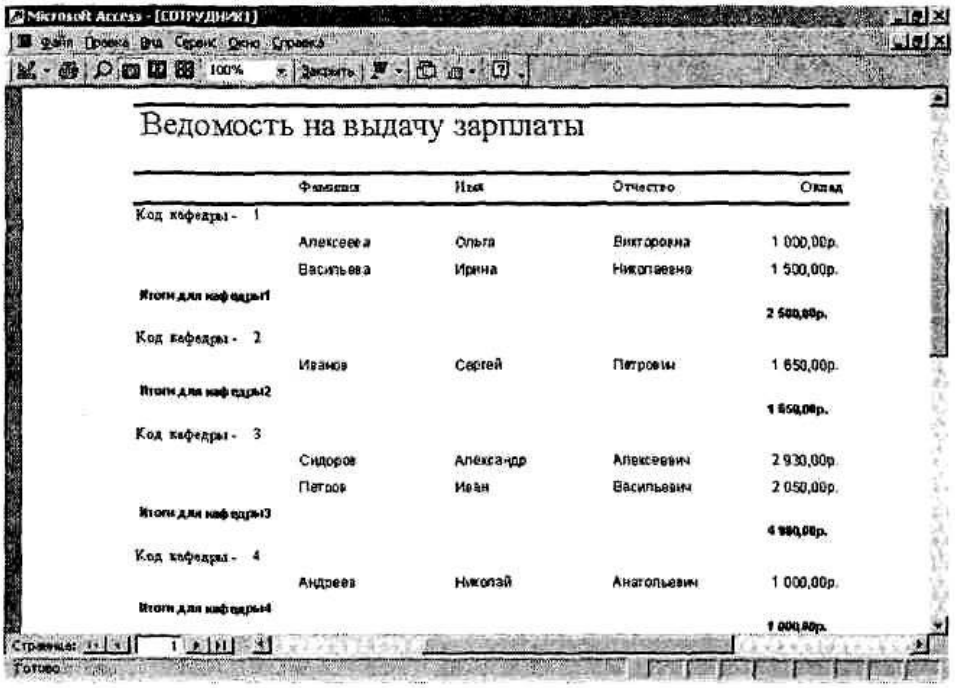


Рис. 9.20. Вид скорректированного отчета
в режиме предварительного просмотра (шаг 2)

Вычисления в отчете

В отчетах (так же как и в запросах, формах) можно использовать вычисляемые поля. Мы уже при рассмотрении отчета, полученного с использованием Мастера, встречались с ними и даже корректировали их (речь идет об итоговых показателях, номерах страниц, дате). Теперь рассмотрим, как можно создавать вычисляемые поля. Поскольку вычисляемое поле, как следует из его названия, является полем, то для включения в отчет следует использовать элемент-поле. Выбрав этот элемент на панели элементов, необходимо позиционировать курсор на нужном месте отчета. При этом в отчете создается элемент, включающий две части - имя поля и значение поля. Внутри прямоугольника, соответствующего имени поля, написано «Имя N», а внутри прямоугольника, соответствующего значению поля, написано слово «Свободный». Чтобы ввести выражение для вычисления значения поля, следует выделить соответствующий элемент, щелчком правой кнопки мыши вывести на экран контекстно зависимое меню, выбрать в нем строку Свойства, в появившемся окне Поле перейти на вкладку **Данные** и в строке **Данные** ввести требуемое выражение. Оно может вводиться вручную либо строиться с использованием построителя выражений. Выражение, вводимое в поле, должно начинаться со знака равенства. Для того чтобы воспользоваться построителем, следует щелкнуть по кнопке с многоточием. Построение выражения выполняется как обычно.

Предположим, что мы хотим ввести в нашу ведомость графу «Подоходный налог» (будем считать, что все сотрудники платят налог в размере

12%). Выражение, записанное в строку «Данные» окна **Свойства полей**, будет иметь вид

= [Оклад] * 0,12.

Чтобы закончить оформление вводимого элемента, перенесем в зону верхнего колонтитула имя этого поля и изменим его название на «Подоходный налог». Ведомость в режиме конструктора теперь имеет вид, представленный на рис. 9.21,а в режиме просмотра - на рис. 9.22.

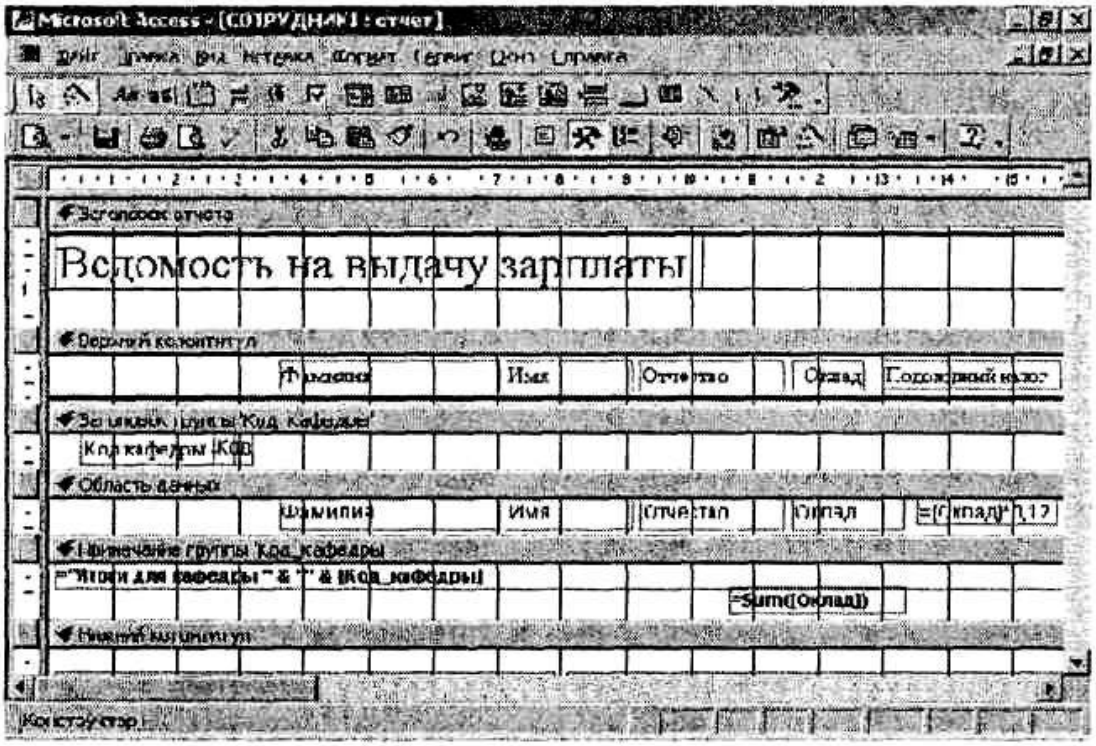


Рис. 9.21. Вид скорректированного отчета в режиме **Конструктор** (шаг 3)

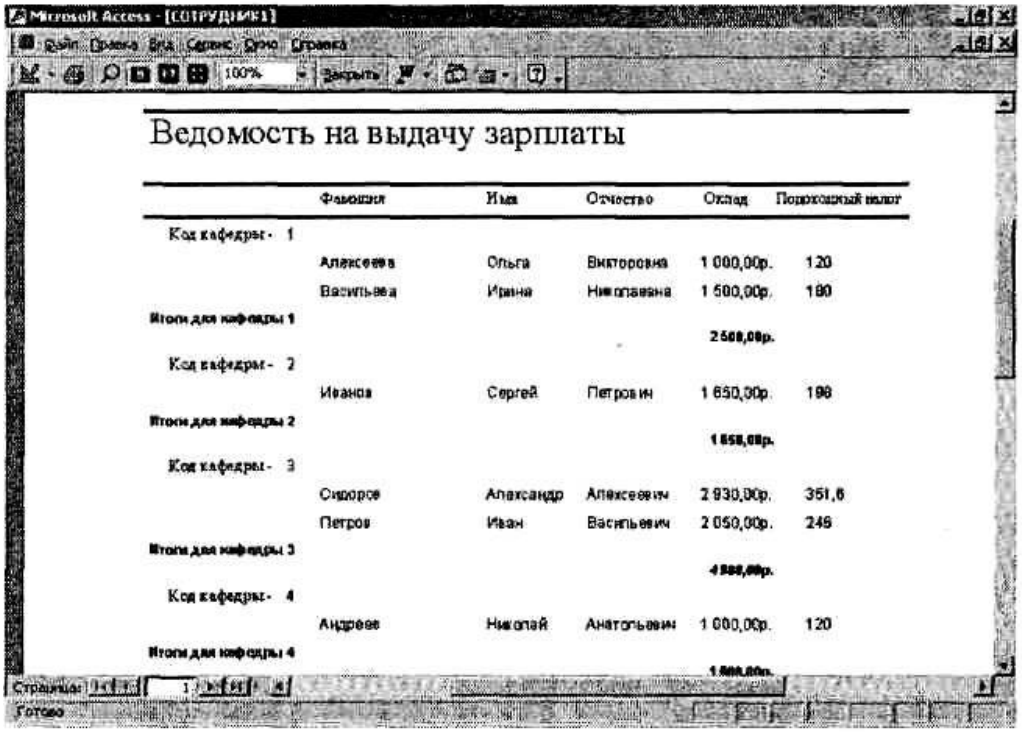


Рис. 9.22. Вид скорректированного отчета в режиме предварительного просмотра (шаг 3)

Несмотря на то что, как мы видели, имеется возможность вводить вычисляемые поля непосредственно в отчет, часто бывает более удобно получить вычисляемое поле в запросе и использовать этот запрос в качестве источника данных для получения отчета.

Нередко в отчетах требуется осуществить нумерацию выводимых данных («Номер по порядку»). Для этого может быть использован следующий прием. Создадим в области данных отчета поле, в свойстве «Данные» которого занесем выражение =1, а в свойстве «Сумма с накоплением» укажем параметр «Для вещего», если необходимо осуществить сквозную нумерацию всех детальных строк отчета, и «Для группы», если в каждой группе нумерация должна начинаться заново. Мы выбрали для нашего примера последнюю из перечисленных возможностей (рис. 9.23). Далее в зоне верхнего колонтитула введем имя этой колонки. После этого отчет в режиме конструктора примет следующий вид (рис. 9.24).

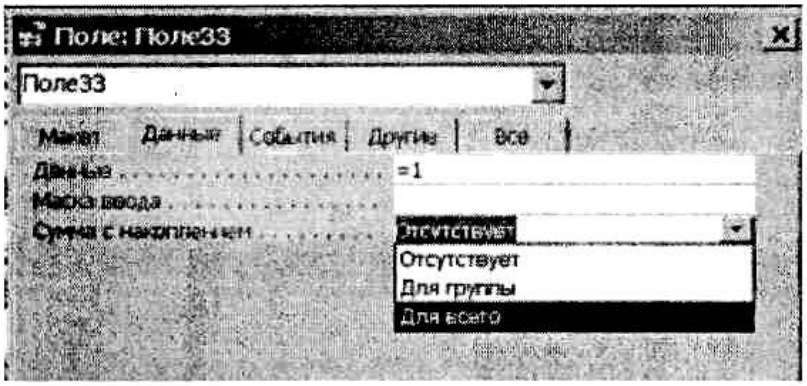


Рис. 9.23. Формирование значения поля «Номер по порядку»

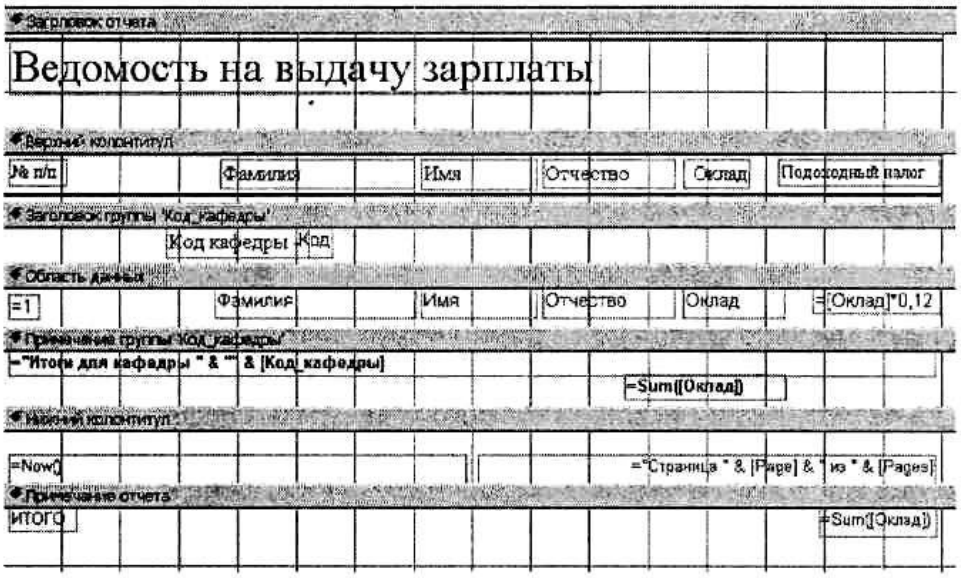


Рис. 9.24. Вид отчета в режиме **Конструктор**
с введенной графой «№ п/п»

Ввод нового поля в отчет

Если возникла необходимость дополнительно ввести в отчет какое-то поле из таблицы/запроса, то это можно сделать несколькими способами. Можно пойти тем же путем, что и при создании вычисляемого поля: создать с помощью панели инструментов элемент-поле и ввести (или выбрать из соответствующей папки при использовании построителей) в него название поля, которое является элементом-источником данных. Но удобнее воспользоваться другой возможностью: вывести на экран список полей источника данных (щелчком по соответствующей кнопке на панели «Конструктор отчетов») и перетащить в выбранную область проектируемого отчета нужное поле из него.

При использовании последнего способа в области данных появляются сразу два элемента: левый — это элемент-подпись, а правый - это собственно создаваемое поле. В элементе-подписи выводится текст, указанный в свойстве «Подпись» для соответствующего поля в таблице или запросе. Если вы вводите элемент в область данных табличного документа, то следует отдель-

но выделить подпись, вырезать ее и вставить в нужное место верхнего колонтитула.

Группировка

Мы уже рассматривали возможности группировки данных, когда получали отчет с использованием Мастера. Посмотрим, как это можно сделать с использованием **Конструктора**. Для этого можно воспользоваться кнопкой на панели «Конструктор отчетов» или выбрать соответствующую позицию из меню **Вид**, в появившемся окне выбрать то поле, которое будет использоваться для группировки, и задать нужные значения свойств «Верхний колонтитул» и «Нижний колонтитул» (рис. 9.25).

Предположим, что мы хотим в нашей ведомости сделать дополнительную группировку по полю «Пол» и сосчитать среднюю зарплату по этой подгруппе. Предварительно введем это поле в отчет, после чего выберем это поле для группировки и зададим значение «**Да**» для «Верхнего колонтитула» и «Нижнего колонтитула». Следует обратить внимание, что расположение элементов в окне **Сортировка и группировка** имеет большое значение. Оно определяет старшинство сортировки/группировки.

Для получения желаемого результата поместим поле «Пол» в данном списке на вторую позицию. В итоге в отчете появятся две дополнительные зоны. В зону заголовка группы «Пол» перенесем подпись соответствующего поля и само поле, а в примечание группы «Пол» включим вычисляемое поле, отражающее среднюю зарплату. Отчет в режиме **Конструктор** теперь имеет вид, представленный на рис. 9.26. Фрагмент этого документа в режиме предварительного просмотра приведен на рис. 9.27.

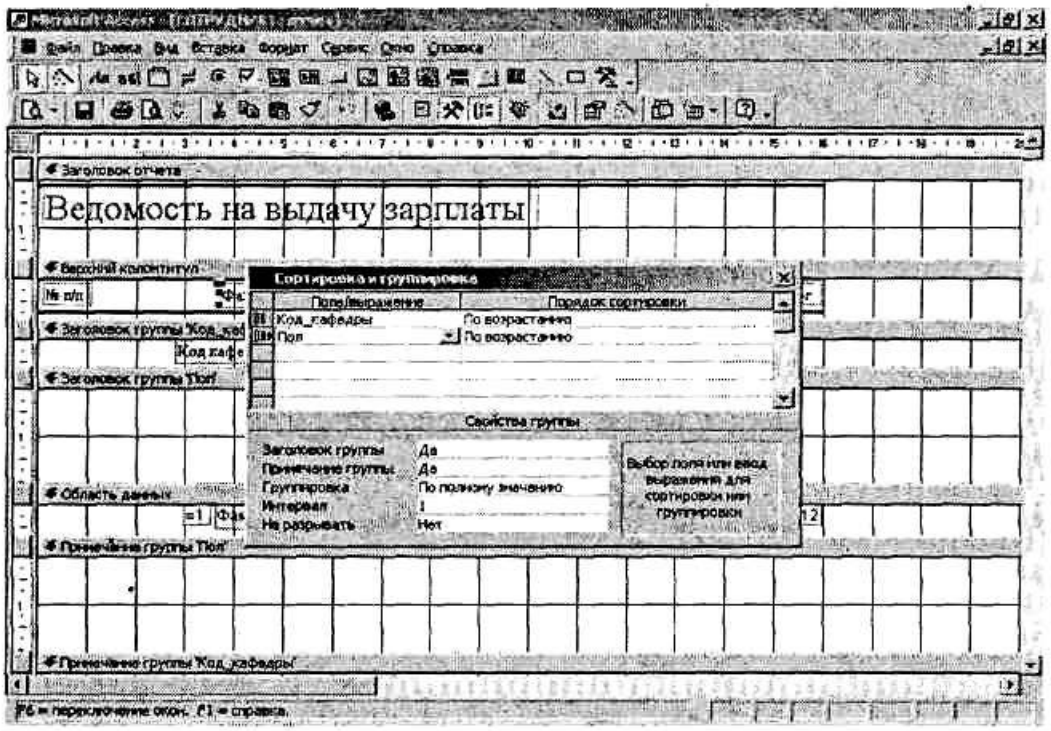


Рис. 9.25. Введение уровней сортировки и группировки

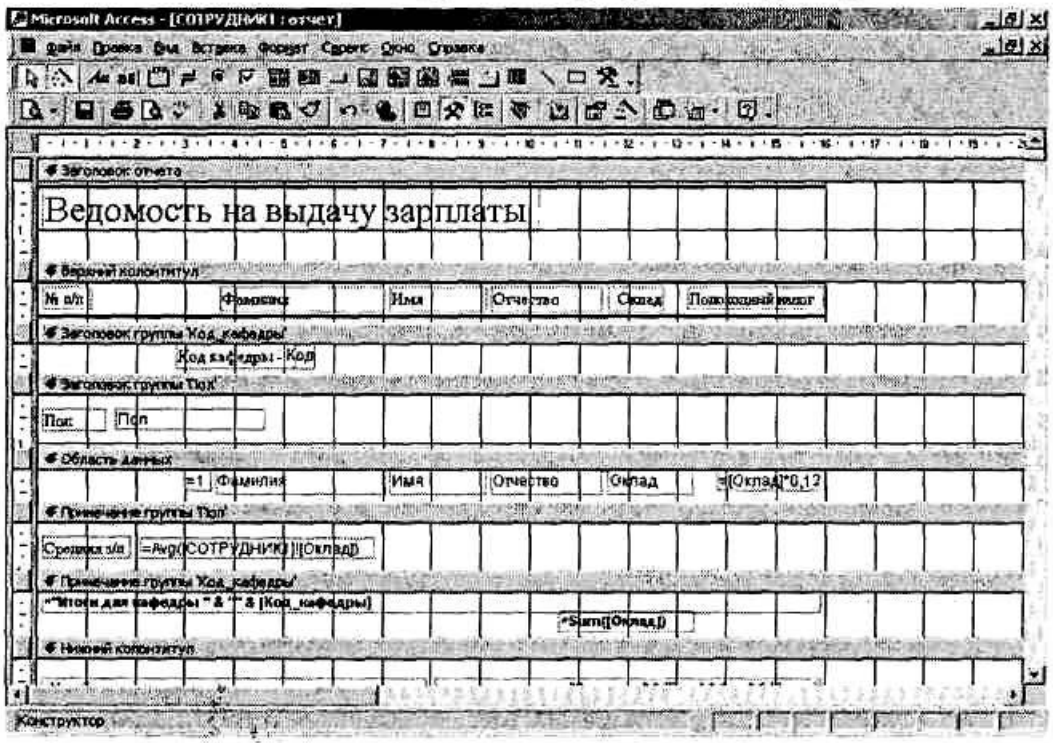


Рис. 9.26. Вид скорректированного отчета в режиме **Конструктор** (введение дополнительного уровня группировки)

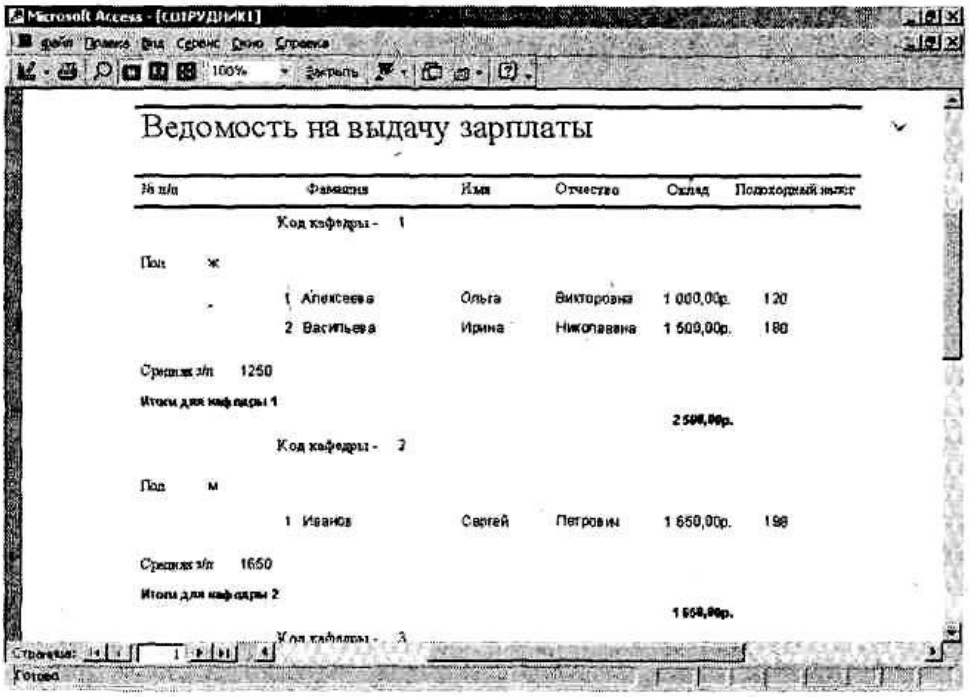


Рис. 9.27. Фрагмент отчета в режиме предварительного просмотра после введения дополнительного уровня группировки

Всего в отчете может быть задано до 10 уровней группировки.

Использование графических элементов

Для улучшения внешнего вида в отчете можно использовать графику: линии, прямоугольники, графические изображения. Чтобы применять в отчете соответствующие элементы, на панели инструментов следует щелчком мыши выделить соответствующую кнопку:

- линия;
- прямоугольник;
- рисунок;
- свободная рамка объекта;
- присоединенная рамка объекта.

Для изображения линии необходимо установить курсор на место, где должно располагаться начало линии, и при нажатой левой кнопке мыши указать, где будет располагаться конец линии. Отпустив кнопку, вы увидите на экране нужный элемент.

Если вы хотите вставить в документ рисунок, например логотип фирмы, этот рисунок должен быть предварительно создан и сохранен. При выборе кнопки Рисунок высветится окно, в котором можно выбрать файл, содержащий рисунок.

Задание номеров страниц

Для печати номеров страниц в форме или отчете следует использовать свойства «Page» и «Pages». Эти свойства позволяют вывести в нижнем или верхнем колонтитуле отчета номера страниц в формате «Страница п из пп». Свойство «Page» указывает номер текущей страницы при печати формы или отчета, свойство «Pages» - полное число страниц в форме или отчете.

При задании номеров страниц можно воспользоваться следующей последовательностью действий:

1. Открыть отчет/форму в режиме **Конструктор**.
2. Выбрать в меню **Вставка** команду **Номера страниц**.

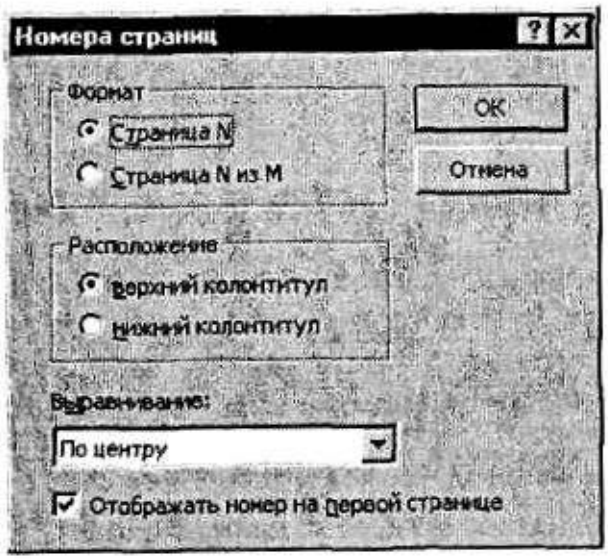


Рис. 9.28. Вид окна Номера страниц

В окне **Номера страниц** (рис. 9.28) выбрать параметры, определяющие формат, расположение и выравнивание номеров страниц.

9.2.4. Создание отчета, базирующегося на нескольких таблицах

В нашем примере в качестве источника для отчета использовалась одна таблица. Часто требуется ввести в отчет данные из нескольких таблиц. Даже в нашем примере желательно выводить в отчете не код кафедры, а ее наименование. Сделать это можно несколькими способами. Во-первых, можно выбрать несколько источников на том шаге создания отчета, на котором выбираются поля в отчет (см. рис. 9.6). Таблицы следует выбирать в порядке их соподчиненности (т.е. по .цепочке 1:М). В нашем случае сначала нужно выбрать таблицу «Кафедра» (рис. 9.29), выбрать поля из этого источника и далее, не переходя к следующему шагу, выбрать таблицу «Сотрудник» и ее поля. В этом случае группировка по кафедре будет создана автоматически. На последующих шагах проектирования отчета можно как отказаться от этого уровня группировки, так и создать новые уровни.

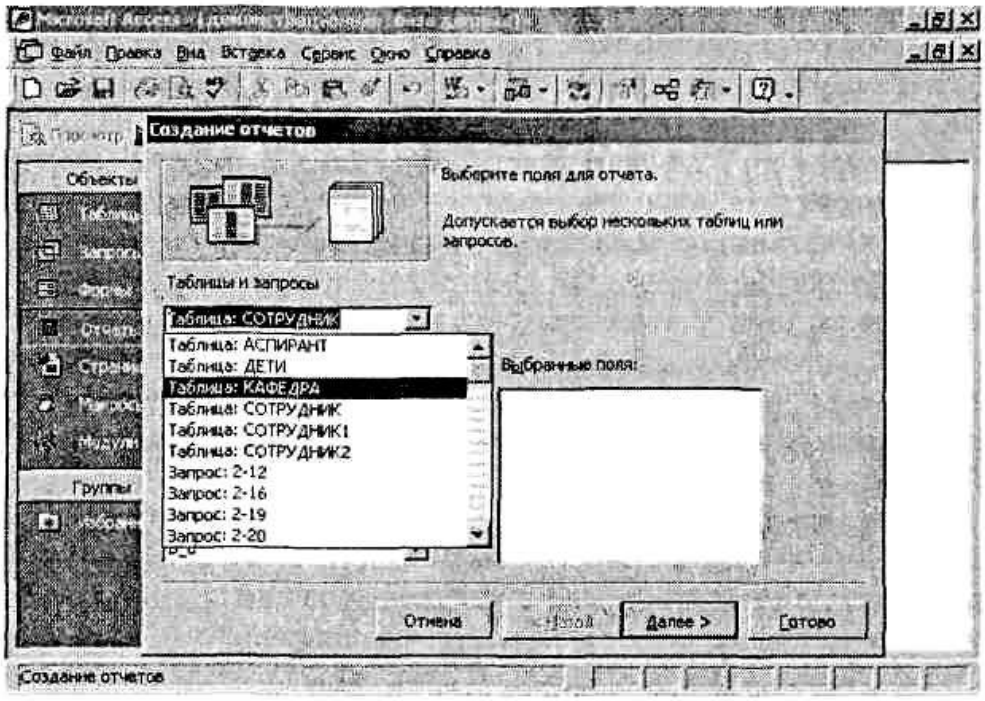


Рис. 9.29. Выбор источников для формирования отчета

Другим способом создания документов, базирующихся на нескольких таблицах, является применение в качестве источника для отчета запроса, использующего несколько таблиц. Во многих случаях этот способ более удобен для создания запросов.

9.2.5. Создание сложных отчетов

В категорию «сложных» в Access отнесены отчеты, включающие в своем составе подчиненные отчеты, перекрестные отчеты и отчеты, печатающиеся в несколько колонок.

Рассмотрим первую из подкатегорий сложных отчетов. *Подчиненным* называют отчет, вставленный в другой отчет. При комбинировании отчетов один из них является главным. *Главный* отчет может быть как *присоединенным*, так и *свободным*, т.е. не базирующимся на таблице, запросе или инструкции SQL.

Свободный главный отчет может служить контейнером нескольких, не связанных между собой отчетов, которые требуется объединить. Например, вы создаете отчет о продажах за определенный период и хотите включить в него разделы, содержащие данные, сгруппированные по разным признакам: по территориальному, по продуктам, по сотрудникам. Эти разделы будут самостоятельными, не соподчиненными.

Главный отчет связывают с таблицей, запросом или инструкцией SQL в тех случаях, когда в него требуется вставить подчиненные отчеты, в которых выводятся данные, связанные с данными в главном отчете. Например, в главном отчете могут быть выведены записи о каждом сотруднике, а в подчиненном - данные о детях каждого сотрудника. В главный отчет наряду с подчи-

ненными отчетами могут включаться также подчиненные формы, причем число таких подчиненных форм не ограничивается. Более того, главный отчет может содержать подчиненные формы или отчеты двух уровней вложенности. Например, в отчете может содержаться подчиненный отчет, который в свою очередь содержит подчиненную форму или подчиненный отчет.

Подчиненные отчеты могут быть получены либо путем создания подчиненного отчета в существующем отчете, либо путем добавления существующего отчета в другой существующий отчет. В последнем случае добавляемый отчет становится подчиненным.

Если подчиненный отчет должен быть связан с главным отчетом, то перед выполнением следующих действий убедитесь, что правильно установлены связи между соответствующими таблицами.

Создать отчет, включающий подчиненные, можно, выполнив определенную последовательность действий.

1. Откройте отчет, который должен быть главным отчетом, в режиме **Конструктор**.
2. Убедитесь, что кнопка **Мастера** на панели элементов нажата.
3. Щелкните по кнопке **Подчиненная форма/отчет** на панели элементов.
4. Установите указатель в отчете на том месте, куда требуется поместить подчиненный отчет, и нажмите кнопку мыши. В результате появится диалоговое окно **Создание подчиненных форм**.
5. Выполняйте инструкции, выводящиеся в диалоговых окнах Мастера.

После нажатия кнопки **Готово** элемент управления «Подчиненная форма/отчет» будет вставлен в главный отчет. Кроме того, будет создан отдельный отчет, выводящийся как подчиненный отчет.

Создать отчет, включающий подчиненные, можно и другим способом, а именно: после открытия отчета, который должен быть главным отчетом, в режиме **Конструктор** нажмите клавишу [F11] для перехода в окно базы данных и переместите с помощью мыши отчет или таблицу из окна базы данных в тот раздел главного отчета, в который требуется поместить подчиненный отчет.

9.2.6. Свойства

Отчет в целом, каждая зона и каждый элемент отчета имеют большой набор свойств. Выйти в окно **Свойства** можно, позиционируясь на соответствующем элементе, щелчком правой кнопки вывести на экран контекстно зависимое меню и выбрать строку «Свойства». При этом на экране появляется соответствующее окно, в котором перечислены все свойства поля. Их можно просматривать по частям, выбирая соответствующие вкладки, либо увидеть одновременно, перейдя на вкладку **Все**.

Свойства, собранные на вкладке Макет, определяют элементы оформления данных:

- размещение поля на листе (от левого края, от верхнего края);
- размеры поля (ширина, высота);

- «внешний» вид поля и выводимых в нем данных (тип фона, цвет фона, оформление, тип границы, цвет границы, ширина границы, цвет текста, шрифт, размер шрифта, насыщенность, курсив, подчеркнутый, выравнивание текста);
- способ представления данных в поле (формат поля, число десятичных знаков, вывод на экран, расширение, сжатие).

Изменение свойств проводится путем выбора из раскрывающегося списка нужного значения. Большинство свойств очевидно и не требует каких-либо пояснений.

Свойства на вкладке **Данные** определяют, что выводится в поле. Использование этих свойств («Данные» и «Сумма с накоплением») мы уже демонстрировали выше.

Свойства, собранные на вкладке **Другие** (имя и дополнительные сведения), используются соответственно для задания имени поля и задания примечаний, относящихся к нему.

Если требуется изменить свойства нескольких полей, то их выделяют при нажатой левой кнопке мыши, а затем указывают необходимые свойства. Они распространяются на все выделенные поля.

9.2.7. Создание отчета анкетной формы

Для создания отчета анкетной формы необходимо выполнить следующие действия:

- в окне базы данных выбрать вкладку **Отчеты**;
- щелкнуть по кнопке **Создать**;
- в диалоговом окне **Новый отчет** выбрать «**Автоотчет: в столбец**»; в этом случае каждое поле образует отдельную строку с заголовком слева.

После того как отчет приобретет нужный вид, его следует сохранить. Для этого можно либо выполнить команду **Файл/Сохранить** как, либо попытаться закрыть отчет и на вопрос о том, следует ли сохранять сделанные изменения, ответить утвердительно, после чего дать отчету имя.

9.2.8. Совместная работа с другими приложениями MS Office

Существуют несколько способов использования данных Microsoft Access в Microsoft Word. Число возможностей зависит от версии используемого программного продукта.

Для связи с Office предназначена специальная кнопка. В контексте данного раздела наибольший интерес представляют возможности получения сложных документов, а не просто операции экспорта/импорта.

Если позиционироваться на какой-либо таблице и щелкнуть по стрелке на кнопке Связи с Office, то высветится список возможных вариантов взаимодействия:

1. слияние с MS Word;
2. публикация в MS Word (создание нового документа и установление

связи с ним);

3. анализ в MS Excel.

Если выбрать вариант 1, то можно создавать документы Word, в которые включаются значения полей из выбранной таблицы. После выбора этого варианта откроется соответствующее окно (рис. 9.30).

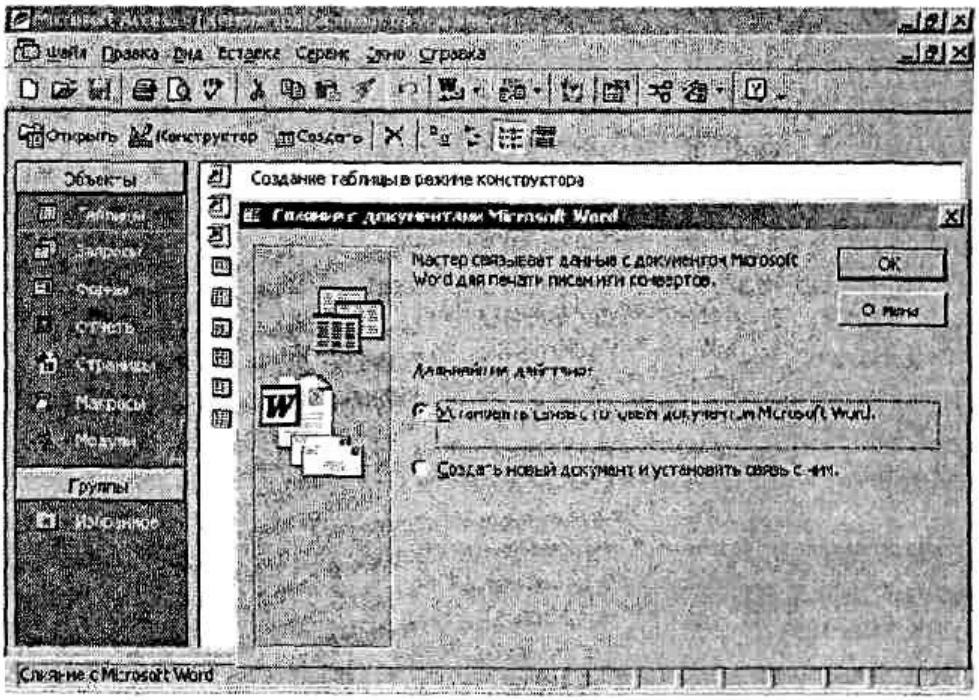


Рис. 9.30. Слияние с документами Microsoft Word

Слияние может осуществляться как с готовым документом Microsoft Word, так и с новым документом.

Предположим, что мы хотим напечатать поздравительные открытки своим сотрудникам. Текст у нас заранее не подготовлен, поэтому выберем второй вариант. После этого открывается документ Word, в котором активна дополнительная панель слияния. Текст, который без изменения будет печататься в каждом документе, набирается как обычно. В том месте документа, куда надо вставить значения поля из таблицы/запроса, следует щелкнуть по стрелке на кнопке «Добавить поле слияния» на панели «Слияние». В открывшемся списке следует выбрать то поле, которое вы хотите вставить в документ. В режиме создания документа в этом месте появится ссылка на выбранное поле.

Мы в нашем поздравлении хотим использовать обращение «дорогой/дорогая», которое зависит от пола сотрудника. Для того чтобы выбиралось нужное обращение, воспользуемся кнопкой «Добавить поле Word» на панели инструментов «Слияние» и в появившемся списке выберем строку «IF...THEN...ELSE». Поле IF обозначает выполнение одной из двух операций, в зависимости от выполнения указанных условий. Нужное нам условие будет выглядеть так, как представлено на рис. 9.31.

Далее в документ необходимо вставить поля слияния, воспользовавшись соответствующей кнопкой. В открывшемся далее окне (рис. 9.32) следует выбрать нужные поля.

Подготовленный нами текст будет иметь вид:

Дорогая «имя» «отчество»!
Поздравляем Вас с наступающим Новым годом!
Счастья! Здоровья!
Успехов в нашей совместной деятельности!

Вместо ссылок при просмотре и печати будут выводиться конкретные имена и отчества сотрудников.

Кроме получения документов слиянием можно, установив связь с MS Word, воспользоваться позицией меню **Сервис/Конверты и наклейки**.

Для получения сложных документов можно также использовать возможность «Публикация в MS Word». Публиковать можно таблицы, запросы, отчеты.

Для публикации соответствующего объекта нужно предварительно позиционироваться на нем в окне базы данных.

Использование возможности «Анализ в MS Excel» приводит к преобразованию таблицы данных или запроса в формат MS Excel.

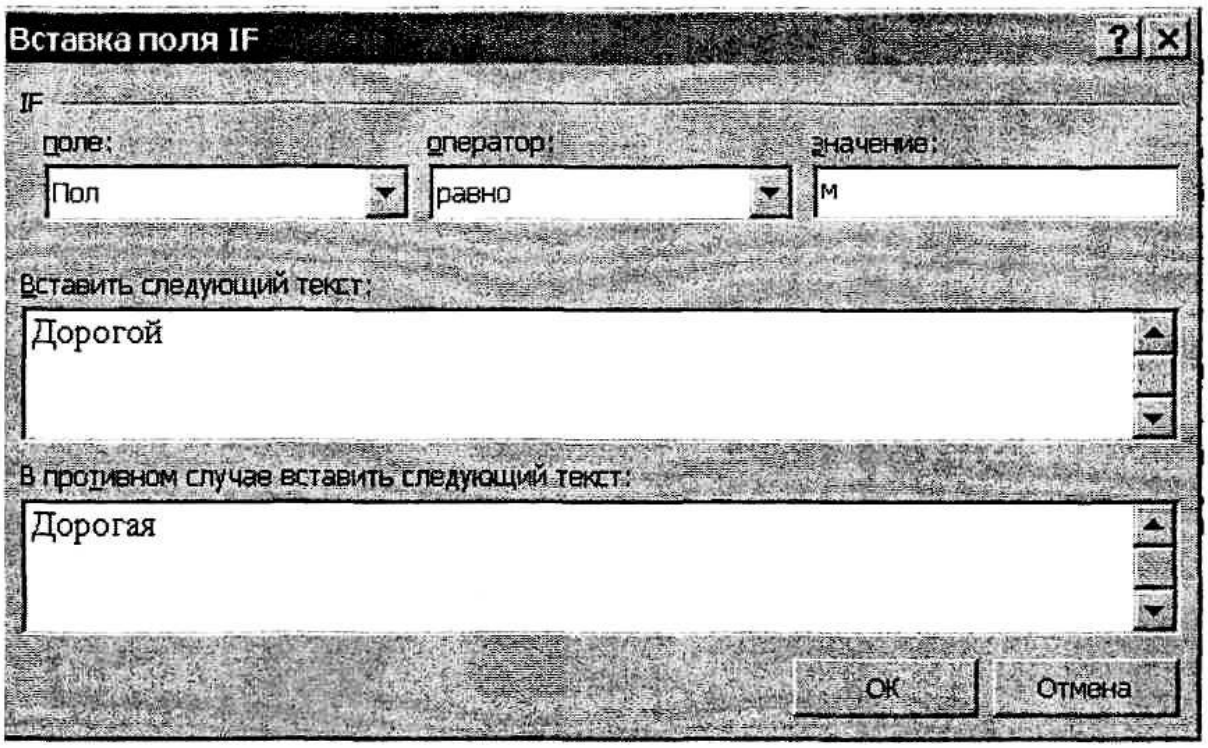


Рис. 9.31. Задание условия при вставке поля

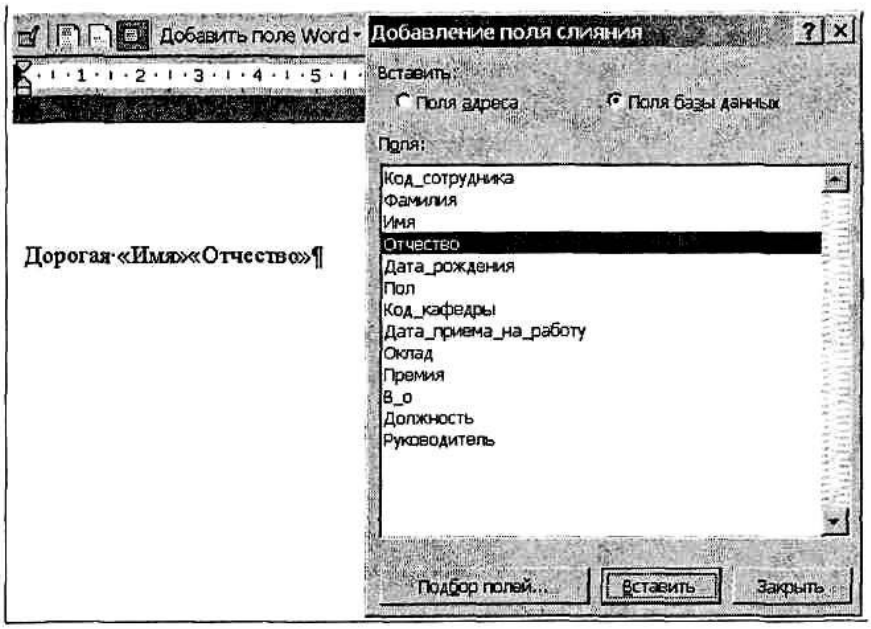


Рис. 9.32. Включение полей слияния

На это следует обратить внимание

- Генераторы отчетов являются мощным средством автоматизации проектирования информационных систем.
- Современные генераторы отчетов позволяют получать документы разной структуры и степени сложности, использовать разные формы представления информации, обладают развитыми возможностями по оформлению документов.
- Генераторы отчетов, включенные в состав конкретных СУБД, различаются по своим функциональным возможностям, удобству использования. Однако принципы, положенные в их основу, являются общими. Знание этих принципов позволяет легко осваивать разные реализации этих систем.

Контрольные вопросы

1. Что такое «отчеты»?
2. Что такое «генераторы отчетов»?
3. Дайте классификацию форм отчетов.
4. Какие возможности предоставляет пользователю генератор отчетов СУБД Access?
5. Что такое макет отчета и какова его структура?
6. Что такое Мастера и какие Мастера включены в генератор отчетов Access?
7. Каков порядок создания отчета?
8. Как можно получить документ табличной формы, анкетной формы?
9. Как можно обеспечить сортировку данных в отчете?
10. Можно ли выводить в документ данные, явно не содержащиеся в таблицах базы данных? Если да, то как это сделать?
11. Как можно получить отчет, содержащий информацию из нескольких

таблиц БД? Какие разновидности отчетов при этом можно получить?

12. Каковы возможности группировки данных в отчете?
13. Сколько степеней итогов позволяет получить генератор отчетов Access?
14. Можно ли получить документ, содержащий только итоговые строки? Если да, то как это сделать?
15. Как пронумеровать страницы отчета?
16. Как пронумеровать строки в отчете?
17. Что собой представляет «Слияние с MS Word»? «Публикация в MS Word»?
18. Какие свойства элементов отчета можно задавать и в каких целях их можно использовать?

Глава 10 РАСПРЕДЕЛЕННЫЕ БАНКИ ДАННЫХ

10.1. Основные понятия

Будем различать понятия «распределенные банки данных» (РБнД) и «распределенные базы данных» (РБД).

РБнД будем называть БнД, в котором распределен хотя бы какой-либо один компонент БнД (не обязательно БД). В такой трактовке информационная система, обеспечивающая многопользовательский доступ к централизованной базе данных, будет считаться одной из разновидностей РБнД. Такое понимание РБнД не является общепринятым.

Распределенная база данных (DDB - Distributed DataBase) - это совокупность множества взаимосвязанных баз данных, распределенных в компьютерной сети. РБД распределена физически, но логически - едина (имеет общую схему данных).

Система управления распределенной базой данных — программная система, позволяющая управлять базой данных таким образом, чтобы ее распределенность была прозрачна для пользователей.

Сервер баз данных - СУБД, основанная на архитектуре «клиент-сервер».

SQL-сервер - собирательный термин, относящийся ко всем серверам баз данных, основанных на SQL. Наиболее популярные SQL-серверы перечислены в табл. 10.1.

Таблица 10.1

СУБД	Производитель	URL
Oracle	Oracle Corp.	www.oracle.com
MS SQL Server	Microsoft	www.microsoft.com
Informix	Informix	www.informix.com
Sybase	Sybase	www.sybase.com
DB2	IBM	www.4.ibm.com

Современным серверным СУБД присущи следующие характерные черты:

- реализация для различных платформ (UNIX, Windows, Linux);
- наличие административных утилит;
- резервное копирование данных;
- обслуживание репликаций;
- параллельная обработка данных в многопроцессорных системах;
- поддержка OLAP и создания хранилища данных (многие производители серверных СУБД поставляют такие серверы отдельно (Oracle, Informix), некоторые включают их в состав сервера реляционных БД (MS SQL Server 7.0 и выше));
- распределенные запросы и транзакции;
- использование средств проектирования БД (универсальных или ориен-

тированных на конкретную СУБД);

- поддержка доступа к данным через Интернет.

К распределенным системам помимо требований, предъявляемых ко всем ИС, выдвигаются дополнительные требования:

- независимость от расположения данных;
- независимость от СУБД, ОС и аппаратного обеспечения;
- максимальная автономия.

При работе в РБнД возникают дополнительные проблемы. Причинами, вызывающими эти проблемы, являются:

- параллельное одновременное обращение пользователей к БД;
- распределение функций по обработке данных между компьютерами;
- распределение данных между компьютерами;
- авторизация пользователей и распределение ролей между ними;
- неоднородность отдельных компонентов.

Для решения этих проблем существуют специальные механизмы, которые будут рассмотрены ниже.

10.2. Классификация РБнД

При классификации РБнД будут рассмотрены только характеристики, присущие именно распределенным ИС. На рис. 10.1 приведена обобщенная схема классификации РБнД.

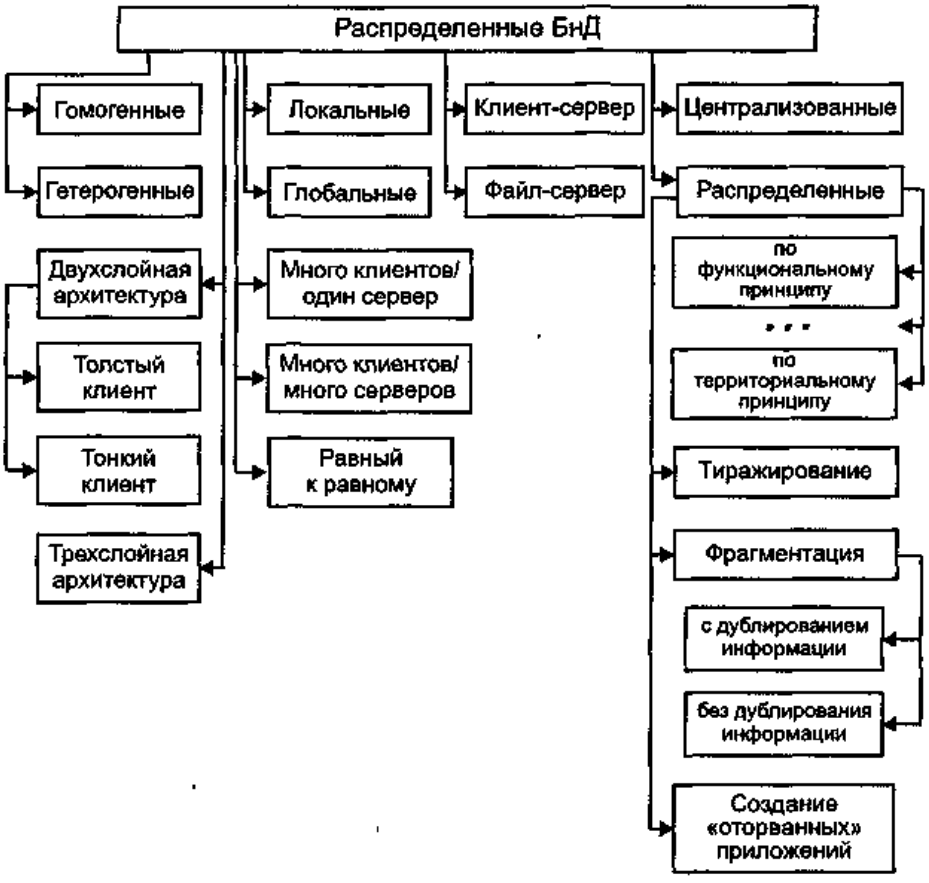


Рис. 10.1. Классификация распределенных банков данных

РБнД реализуются в компьютерных сетях, причем это могут быть как локальные, так и глобальные сети. Вид сети, используемые сетевые операционные системы, безусловно, оказывают существенное влияние на проектирование и функционирование РБнД, но классификация систем по этим признакам выходит за рамки данного учебника.

Как пояснялось при введении понятия РБнД, предполагается, что в РБнД базы данных могут быть как централизованными, так и распределенными. В РБнД с распределенными базами данных используются разные технологии распределения данных по узлам сети. Различают подходы, основанные на фрагментации БД и на тиражировании данных.

При использовании фрагментации единая логическая БД разбивается по каким-либо признакам на составные части (фрагменты), хранящиеся в разных узлах сети.

В РБД могут использоваться разные типы фрагментации. По операциям над таблицами различают горизонтальное, вертикальное и смешанное фрагментирование. Разбиение БД на фрагменты может проводиться по разным признакам. Наиболее часто используются следующие критерии разбиения: территориальный, функциональный, временной.

При использовании технологии *тиражирования* создаются и поддерживаются в согласованном состоянии копии всей БД или ее фрагментов в нескольких узлах сети. Копия базы данных, являющаяся членом набора других копий, которые могут быть синхронизированы между собой, называется *репликой*.

Технология тиражирования по характеру организации данных близка к применению локальных баз данных (так как каждый пользователь использует собственную копию БД), но отличается от децентрализованных систем по способу создания локальных копий и технологии поддержания целостности. Полностью децентрализованная система не предполагает использование специализированных программно-технических средств, поддерживающих целостность БД.

При сравнении децентрализованных и распределенных систем (табл. 10.2) знаком «+» отмечены лучшие значения характеристики.

Таблица 10.2

Характеристика	Система	
	децентрализованная	распределенная
Возможность совместного использования данных		+
Сложность используемых программных средств	+	
Возможность централизованного обеспечения целостности данных		+
Сложность проектирования	+	

Сложность поддержания безопасности	+	
Гибкость и эффективность использования вычислительных ресурсов		+

В РБнД с централизованной БД (много клиентов/один сервер) проблемы управления базой данных решаются относительно просто, поскольку вся она хранится на одном сервере. Задачи, с которыми приходится здесь сталкиваться, - это блокировки при одновременном обращении к данным, управление буферами клиентов и кэширование данных. Управление данными реализуется централизованно на одном сервере.

В системах с распределенной базой данных и проектирование БД, и реализация запросов, и управление системой представляют собой более сложные задачи, но, с другой стороны, такие системы обеспечивают большую гибкость, надежность и быстродействие. В табл. 10.3 приведено сравнение централизованных и распределенных систем.

Таблица 10.3

Характеристика	Система	
	централизованная	распределенная
Сложность используемых программных средств	+	
Уязвимость БД		+
Потери на непроизводительные (накладные) расходы	+	
Ограничение пропускной способности		+
Модульность, возможность расширения системы		+
Сложность проектирования	+	
Сложность поддержания актуальности БД	+	
Сложность поддержания целостности БД	+	
Сложность поддержания безопасности	+	
Возможность уменьшения сетевого трафика		+
Распределение рабочей нагрузки естественным образом на несколько компьютеров		+
Соотношение показателя «стоимость/ эффективность»		+
Требования к аппаратной платформе		+
Сложность администрирования	+	

Используемые технологии распределения данных видоизменяют преимущества и недостатки этих систем. Так, одно из основных преимуществ баз данных, обусловленных интегрированным хранением данных, - сокращение степени дублирования - теряется при использовании технологии тиражирования. Но при этой технологии, в отличие от использования децентрализованных систем, сохраняются возможности контроля целостности данных для системы в целом.

При технологии фрагментации созданные фрагменты могут быть *пересекающиеся* и *непересекающиеся*. Наличие пересекающихся фрагментов делает схожей эту технологию с технологией тиражирования.

На выбор способа организации БД оказывают влияние следующие факторы:

- число пользователей;
- степень пересечения информационных потребностей пользователей;
- объем данных;
- квалификация персонала;
- ограничения (стоимость, время отклика, актуальность информации);
- наличие существующей системы обработки информации и ее характеристика.

Несмотря на то что при использовании распределенных систем используется более сложное и дорогое программное обеспечение, более совершенные технические средства, наибольшие затраты здесь приходятся на эксплуатационные расходы. По данным фирмы Microsoft*, структура затрат в централизованных и распределенных системах выглядит следующим образом:

- в централизованных системах около половины (48%) средств расходуется на аппаратное обеспечение, 14% составляют затраты на программное обеспечение и 28% - на поддержку системы;
- в распределенных системах основные затраты (77%) связаны с поддержкой системы, расходы на аппаратное обеспечение составляют 15%, а на программное обеспечение - только 6% общей стоимости системы.

В распределенных системах расширяется состав и увеличивается объем метаинформации, повышается значимость ее эффективной организации. Могут использоваться разные способы организации метаинформации:

1. *централизованный каталог*;
2. *полностью реплицированный каталог*;
3. *секционированный каталог*;
4. *комбинированный* (1 и 3).

По *способу взаимодействия* функциональных компонентов различают *файл-серверную* и *клиент-серверную* архитектуру РБнД. На рис. 10.2 представлена принципиальная схема обработки данных в архитектуре «файл-сервер». При использовании такой архитектурной модели основная обработка данных проводится на рабочей станции. Такая модель приводит к необходимости передачи больших объемов данных по сети, что увеличивает тра-

* Решения Microsoft'99 (выпуск 7). Microsoft Corporation, 1999//Компьютер-Пресс.

фик, а это, в свою очередь, может привести к замедлению обработки данных, увеличению стоимости, снижению надежности и другим недостаткам.

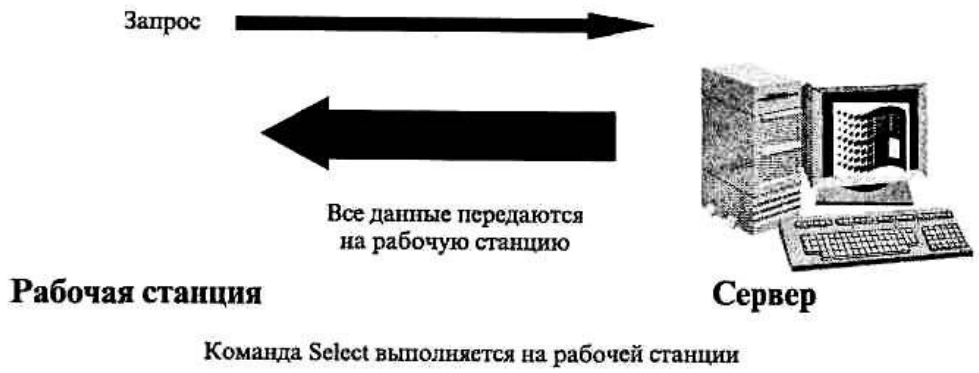


Рис. 10.2. Схема обработки данных в архитектуре «файл-сервер»

В системах с «клиент-серверной» архитектурой (рис. 10.3) основная обработка данных проводится на сервере.

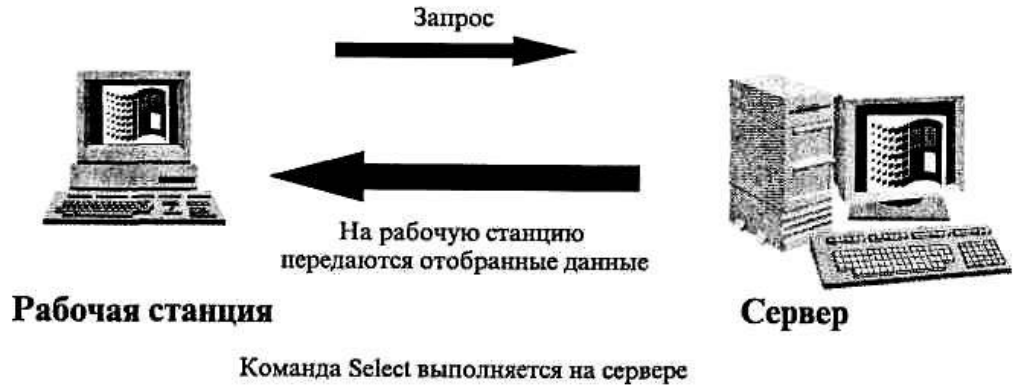


Рис. 10.3. Схема обработки данных в архитектуре «клиент-сервер»

«Клиент-серверные» системы имеют следующие преимущества:

- снижение сетевого трафика за счет выполнения запросов на сервере;
- оптимизация выполнения запросов;
- возможность хранения бизнес-правил на сервере (ограничения целостности, хранимые процедуры, отражающие логику обработки);
- возможность использования CASE-средств для генерации кодов серверных объектов (триггеров, хранимых процедур, текстов SQL-запросов);
- управление пользовательскими привилегиями и правами доступа;
- широкие возможности резервного копирования и архивации данных.

Сравнительные характеристики технологий «файл-сервер» и «клиент-сервер» приведены в табл. 10.4.

Таблица 10.4

Характеристика	«Файл-сервер»	«Клиент-сервер»
Интенсивность сетевого трафика		+
Обеспечение целостности данных		+
Обеспечение безопасности данных		+
Устойчивость к сбоям		+
Сложность проектирования	+	
Сложность эксплуатации системы	+	
Ограничения на число пользователей		+

При обработке данных в сетевой среде выделяют следующие основные группы выполняемых функций:

- презентационная логика (Presentation Layer - PL);
- бизнес-логика (Business Layer - BL);
- логика доступа к ресурсам (Access Layer - AL).

По х*арактеру распределения функций между клиентом и сервером* различают системы с тонким клиентом, толстым клиентом и системы с *трехслойной (трехуровневой) архитектурой*.

Модель с *тонким* клиентом стала активно использоваться в корпоративной среде в связи с распространением интернет-технологий, и в первую очередь Web-браузеров. В этом случае клиентское приложение обеспечивает реализацию PL, а сервер объединяет BL и AL.

Модель с толстым клиентом наиболее часто встречается в уже внедренных и активно используемых системах. Такая модель подразумевает объединение в клиентском приложении как PL, так и BL. Серверная часть при описанном подходе представляет собой сервер баз данных, реализующий AL. К описанной модели часто применяют аббревиатуру RDA - Remote Data Access.

В модели с *трехуровневой архитектурой* физически выделяется «сервер бизнес-логики», на котором и выполняются пользовательские приложения (блок BL).

Как видно из вышеизложенного, существует множество разнообразных технологий работы в распределенной среде.

Некоторые из них представляют собой реализацию на новом технологическом уровне ранее использовавшихся подходов к организации обработки информации.

РБнД могут быть реализованы на *однородных* элементах (*гомогенные системы*) или на *разнородных* (*гетерогенные системы*). Поскольку процесс создания информационной системы практически непрерывен, то обычно эти системы являются гетерогенными. Разнородными могут быть ЭВМ, ОС, СУБД.

Для обеспечения возможности работы в разнородной среде используются разнообразные категории программных средств:

- собственные сетевые драйверы (native software drivers);
- шлюзы (gateways);
- промежуточное программное обеспечение (middleware).

10.3. Транзакции

10.3.1. Понятие транзакции

Транзакция представляет собой законченную совокупность действий над БД, которая переводит ее из одного целостного в логическом смысле состояния в другое.

Понятие *транзакция* широко используется в АИС и может относиться не только к распределенным банкам данных. Однако в РБнД оно становится особенно значимым.

К транзакциям предъявляется набор требований, известный под названием ACID (Atomicity, Consistency, Isolation, Durability). Эти требования вытекают из определения транзакции.

Атомарность (atomicity). Транзакция представляет собой некоторый набор законченных действий. Система обеспечивает их выполнение по принципу «все или ничего» - либо выполняются все действия, тогда транзакция «фиксируется»; либо, если возможность выполнить все действия отсутствует, например в случае сбоев, транзакция «откатывается» назад, а БД остается в исходном состоянии.

Согласованность (consistency). Предполагается, что в результате выполнения транзакции система переходит из одного корректного состояния в другое.

Изолированность (isolation). При выполнении транзакции данные могут временно находиться в несогласованном состоянии. Такие данные не должны быть видны другим транзакциям, пока изменения не будут завершены (т.е. пока все модификации не будут формально зафиксированы). Система обеспечивает каждой транзакции иллюзию того, что та выполняется изолированно, как если бы прочие транзакции либо завершились до ее начала, либо начнут выполняться после ее завершения.

Долговечность (durability). Если транзакция зафиксирована, то ее результаты должны быть долговечными. Новые состояния всех объектов сохранятся даже в случае аппаратных или системных сбоев.

Конкретные СУБД используют различные механизмы управления транзакциями. Некоторые СУБД для задания транзакции используют операторы BEGIN TRANSACTION-END TRANSACTION, и все команды, заключенные между ними, составляют транзакцию. В некоторых системах считается, что, инициируя сеанс работы с SQL, пользователь начинает транзакцию, которая будет продолжаться, пока не будет введен оператор COMMIT WORK, который сделает все изменения, проведенные в ходе транзакции, постоянными,

или оператор ROLLBACK WORK, который отменяет все сделанные в транзакции изменения. После каждого оператора COMMIT или ROLLBACK начинается новая транзакция.

Во многих СУБД присутствует специальный параметр AUTO-COMMIT, который, находясь во включенном состоянии, приводит к автоматической фиксации изменений каждой нормально завершенной операции.

Транзакция может быть помечена как «только чтение» (READ ONLY). При выполнении такой транзакции попытка провести изменение данных будет вызывать сообщение об ошибке. Задание признака READ ONLY позволяет увеличить производительность как этой, так и параллельно исполняемых транзакций.

Существуют многочисленные модели транзакций, обеспечивающие соблюдение требований ACID. Ниже описаны некоторые из них.

10.3.2. Плоские транзакции

Модели плоских транзакций соответствует один управляющий слой, которому подчинено произвольное число элементарных действий.

Согласно правилам обработки плоских транзакций либо должны успешно завершиться все компоненты глобальной транзакции, либо не должна завершиться ни одна из них. Если неудачей завершилось изменение хотя бы одной удаленной базы данных из множества, то и все остальные компоненты должны быть возвращены в состояние, предшествовавшее началу транзакции. Учитывая количество узлов в сети крупной или даже средней организации, можно предположить, что вероятность отказа хотя бы одного узла весьма высока. Если применяется модель плоских транзакций, то придется заново выполнять все составные части транзакции, что существенно повышает требования к вычислительным ресурсам и отнимает значительную долю пропускной способности системы.

10.3.3. Контрольные точки

Модификация модели плоских транзакций, сохраняющая свойство атомарности, но снижающая потребность в повторном выполнении действий, основывается на понятии контрольных точек.

Контрольные точки устанавливаются в прикладной программе для того, чтобы отметить моменты, начиная с которых можно продолжить вычисления в случае возникновения проблем.

По достижении очередной контрольной точки в транзакции создается новое атомарное действие, которое запускается на выполнение. Только последнее атомарное действие всей последовательности может выполнить фиксацию (COMMIT WORK) транзакции; оператор COMMIT WORK передается всем предыдущим атомарным действиям, пока все они не будут зафиксированы. Контрольная точка не приводит к необратимой фиксации выполненной до этого момента работы.

Откаты (ROLL BACK) транзакции могут инициироваться из любого атомарного действия, а не только из последнего, хотя в любой заданный момент времени прерывание может инициировать только действие, запущенное последним. Это значит, что если для какого-то атомарного действия была достигнута контрольная точка, то для этого действия уже не может быть в дальнейшем принято решение об откате. Откат может быть проведен до любой из предыдущих контрольных точек, поэтому менеджер обработки транзакций должен воспринимать параметр, указывающий, до какой именно контрольной точки нужно провести откат.

10.3.4. Многозвенные транзакции

Модель многозвенных транзакций концептуально подобна модели контрольных точек, но она предполагает фиксацию части работы, сделанной до некоторого момента; возможность отката зафиксированных действий исключается. Приложение тем не менее остается в состоянии транзакции, т.е. при этом не происходит инициации очередной транзакции, ее завершения, инициации следующей и ее завершения и т.д. В рамках многозвенной транзакции сохраняются все необходимые элементы контекста выполнения (курсоры баз данных, открытые файлы и т.д.), хотя ресурсы, ставшие ненужными, можно освобождать.

Модель многозвенных транзакций включает оператор CHAIN WORK - неделимую комбинацию операторов BEGIN WORK и COMMIT WORK, которая неравноценна последовательному выполнению операторов BEGIN WORK и COMMIT WORK по отдельности. При выполнении этих операторов по отдельности контекст пропадает; некоторая другая транзакция может вклиниться и изменить значения в базе данных, которые нужны для выполнения следующего звена многозвенной транзакции, прежде чем это звено начнет выполняться. Таким образом, многозвенные транзакции концептуально эквивалентны транзакциям с контрольными точками с той разницей, что откат может проводиться только до последней зафиксированной точки, а не до любой предыдущей контрольной точки.

Обе модели транзакций - многозвенные и с контрольными точками - позволяют описывать последовательность действий; различия касаются лишь возможностей отката и устойчивости выполненных до заданной точки действий.

10.3.5. Вложенные транзакции

Модель вложенных транзакций включает понятие головной транзакции, которая управляет выполнением всей иерархии. В разках иерархии могут присутствовать транзакции разных уровней вложенности. Концевые узлы иерархии представляют собой плоские транзакции. Отдельные ветви иерархии могут иметь разную длину, т.е. концевые транзакции могут находиться на

разном расстоянии от головной транзакции (корня дерева транзакций). Вложенные транзакции называются также *гнездующимися*.

10.4. Проблемы параллелизма и пути их решения

10.4.1. Параллелизм

При параллельном выполнении операций над базой данных могут возникать некоторые проблемы. Одна из них - *проблема утраченных* (потерянных) *обновлений* (Lost update) - заключается в том, что если пользователи параллельно обновляют одни и те же данные, то запомненным будет то обновление, которое было проведено последним. Остальные обновления будут потеряны (рис. 10.4).

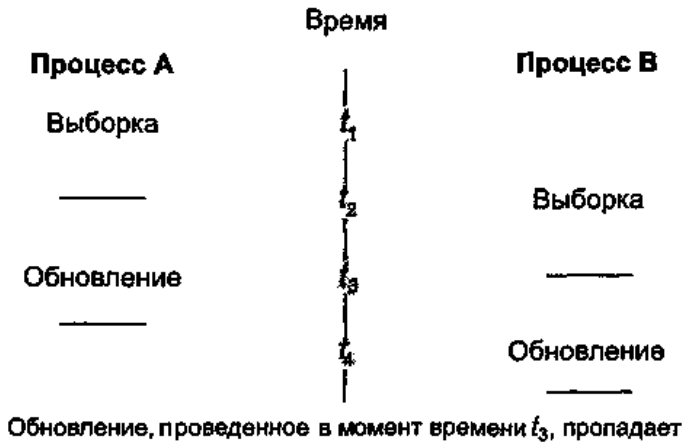


Рис. 10.4. Иллюстрация проблемы утраченного обновления

Другая проблема - *зависимость от незафиксированных обновлений* - состоит в том, что пользователь А может увидеть данные, которые уже были обновлены пользователем В, но эти обновления еще не были окончательно зафиксированы. Далее пользователь В может в силу [различных причин, например из-за выявленных ошибок ввода, провести откат базы данных в исходное состояние (рис. 10.5). Пользователь А в этом случае будет предпринимать действия над ошибочными (данными. Иногда для такого рода проблем используется термин *преждевременное чт*ение (Dirty read).

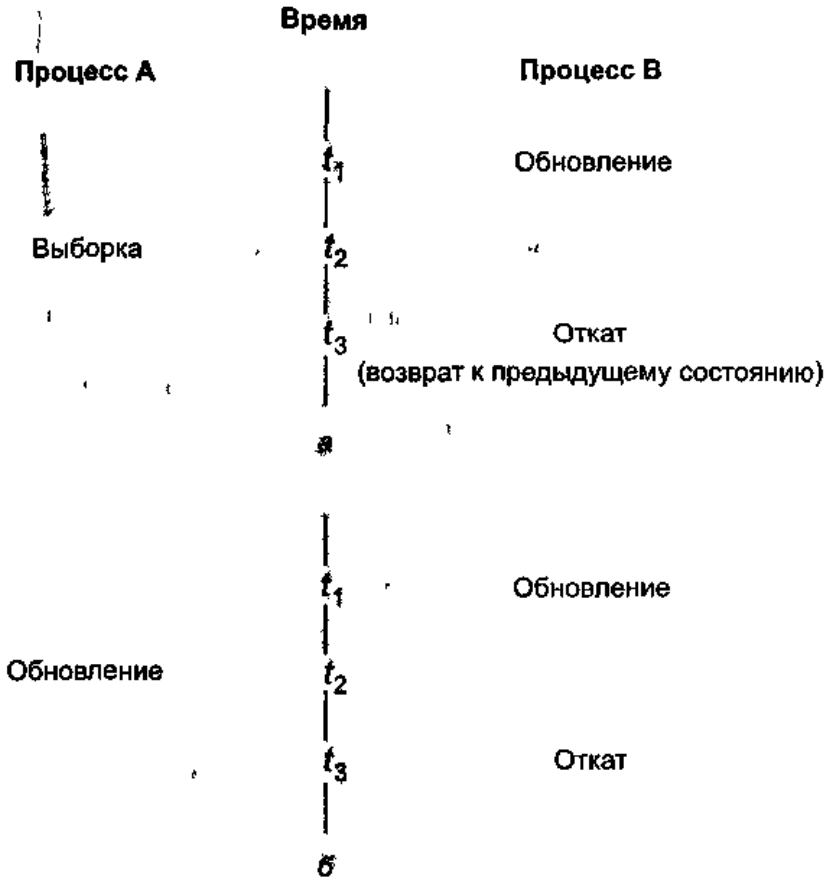


Рис. 10.5. Зависимость от незафиксированных обновлений:
а – преждевременное чтение, *б* – фантомная вставка

Еще одна проблема может возникнуть, если пользователь проводит какую-то групповую обработку данных, не связанную с корректировкой данных, например вычисляет сумму или среднюю величину, а какие-то значения обрабатываемого множества в этот момент претерпевают изменения в результате выполнения параллельной транзакции. Иногда разделяют ситуации, когда проводится изменение существующих записей и когда осуществляется вставка новой записи. Первая проблема называется неповторяющееся чтение, а вторая - фантомная вставка.

Эта ситуация не приводит к искажению информации в базе данных и поэтому в некоторых ситуациях считается допустимой, Например в случае, если специалист проектирует форму отчета и в этом процессе получает черновые отчеты.

10.4.2. Блокировки

Проблемы, возникающие при одновременном обращении, нуждаются в своем разрешении. Наиболее популярные алгоритмы управления одновременным доступом основаны на механизме блокировок. *Блокировка* заключается в запрещении некоторых операций над данными (чаще - корректировки информации), если ее обрабатывает (корректирует) другой пользователь. В

такой схеме всякий раз, когда транзакция пытается получить доступ к какой-либо единице данных, на эту единицу накладывается блокировка.

Обобщенная схема классификации блокировок приведена на рис. 10.6.

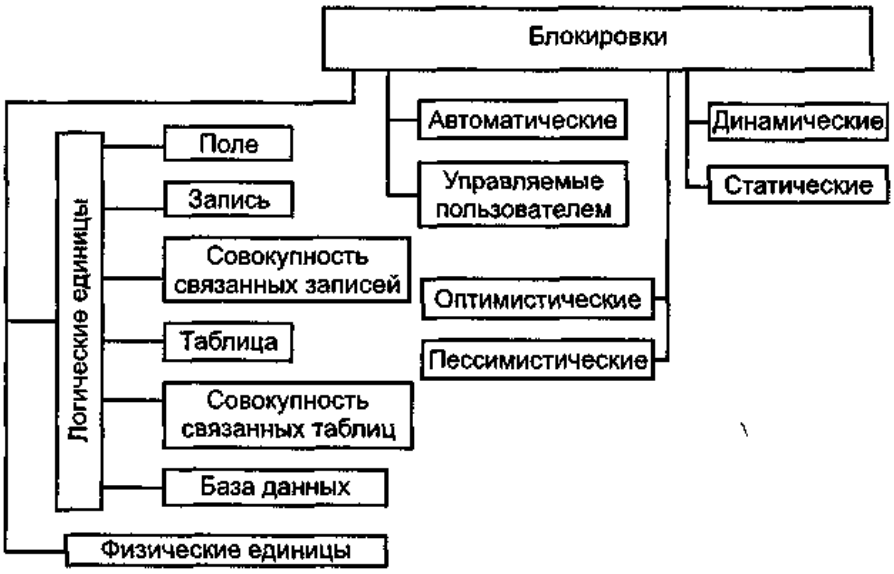


Рис. 10.6. Классификация блокировок

Блокировки накладываются в соответствии с правилами совместимости блокировок, исключающими конфликты *чтение-запись*, *запись-чтение* и *запись-запись*. Сериализуемость транзакций заведомо гарантируется, если блокировки, относящиеся к одновременно выполняемым транзакциям, удовлетворяют следующему правилу: «Ни одна блокировка от имени какой-либо транзакции не должна устанавливаться, пока не будет снята ранее установленная блокировка». Это правило известно под названием *двухфазового блокирования*, поскольку транзакция проходит при этом сначала фазу роста, когда она устанавливает блокировки, а затем фазу сжатия, когда блокировки снимаются. В общем случае снятие блокировок до завершения транзакции проблематично, поэтому в большинстве алгоритмов управления одновременным доступом применяется подход, когда блокировки не снимаются до конца транзакции.

Блокировка может выполняться *автоматически*, а может и управляться *пользователем*. Включение автоматической группировки обусловливается выполняемой над данными операцией. Желательно, чтобы как можно больше ответственности за блокировку было перенесено с разработчика или пользователя на систему управления базой данных.

В зависимости от блокируемых информационных единиц можно выделить следующие *уровни блокирования: база данных, совокупность связанных таблиц, таблица, совокупность связанных записей, запись, поле*. Выше были названы логические единицы реляционных баз данных.

Бывает, что уровни блокирования определяются в терминах физических единиц информации. Уровень блокирования может зависеть не только от СУБД, но и от операционной системы и даже от архитектуры компьютера. В этом случае необходимо знать специфику конкретной платформы, на кото-

рой реализована система. Объектами блокирования могут являться страница (page), группа страниц, область базы данных (dbspace) и др. Терминология и реализация в значительной степени зависят от конкретной системы.

Физическая единица может включать как часть таблицы, так и несколько разных таблиц. Все это скажется на частоте возникновения конфликтов и времени обработки, что необходимо учитывать при физическом проектировании БД и при управлении параллельным доступом.

В конкретных СУБД могут быть реализованы не все, а только некоторые из перечисленных уровней блокировок. Так, практически нигде не реализована блокировка на уровне поля. В Microsoft SQL Server только начиная с версии 6.5 был добавлен уровень блокировки записи, и то только для операций типа INSERT. В dBase, FoxPro предусматривается блокировка на уровне таблиц и записей.

Блокировка на нижних уровнях приводит к перегрузке менеджера блокировок и, как следствие, к падению производительности системы. С другой стороны, блокировка на более высоком уровне мешает конкурирующим пользователям получить доступ к нужным данным.

Некоторые системы предусматривают *динамическую* схему блокировки, заключающуюся в том, что сначала транзакция блокирует большую информационную единицу, например страницу. Если проявляется другая транзакция, претендующая на какую-то запись внутри этой страницы, то первая транзакция автоматически уменьшит зону блокировки до уровня записи.

Различают *пессимистические* и *оптимистические* блокировки. Пессимистические блокировки запрещают доступ к данным других транзакций, когда какая-то транзакция уже работает с ними. Последующие операции, которые могут привести к конфликту, либо ставятся в очередь, либо отменяются.

Оптимистические блокировки разрешают параллельное выполнение транзакций, отслеживают случаи возникновения конфликтов и обеспечивают их разрешение.

10.4.3. Режимы доступа к информации

При работе в многопользовательской среде файлы могут быть открыты в одном из режимов - разделяемом или исключительном. При исключительном (монопольном, эксклюзивном) режиме доступа с данной информационной единицей может работать только тот пользователь, который первый открыл файл. Эксклюзивное использование иногда называют блокировкой типа X (eXclusive lock), а разделяемое (блокировка с взаимным доступом) - S (Shared lock). Исключительные блокировки используются для операторов, изменяющих структуру таблицы или значения тех или иных полей.

Возможные сочетания видов блокировок приведены ниже.

	X	S	-
X	N	N	Y
S	N	Y	Y
-	Y	Y	Y

Эти же виды блокировок могут быть использованы и при блокировании других информационных единиц, например записи. Выбор вида (блокировки и информационной единицы, к которой она относится, зависит от того, какая операция выполняется, как много таких операций, каковы ограничения по времени выполнения обработки. Например, если осуществляется массовая корректировка какого-либо файла, то лучше открыть его в эксклюзивном режиме, чем последовательно блокировать каждую запись.

На рис. 10.7 изображен механизм использования блокировок при выполнении параллельных операций над данными. На этом рисунке представлена тупиковая ситуация, когда две транзакции одновременно ждут снятия блокировки. СУБД должны иметь механизмы разрешения тупиковых ситуаций.

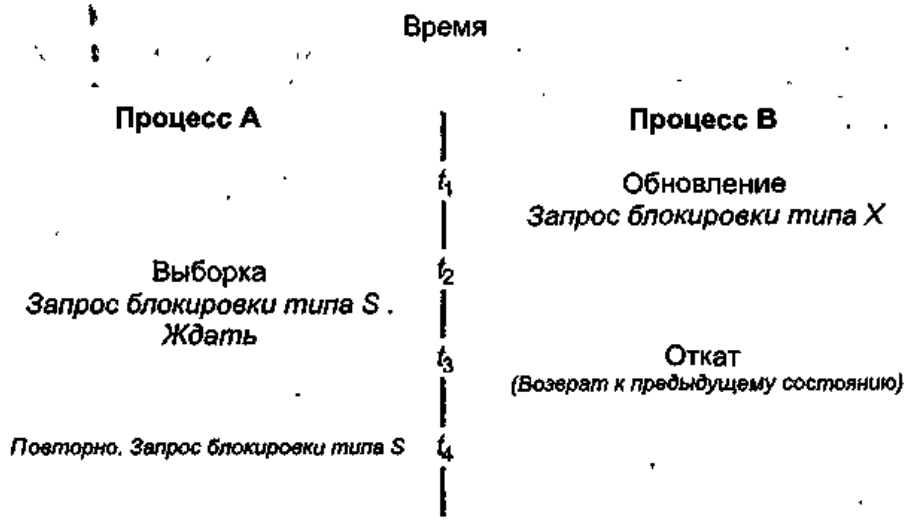


Рис. 10.7. Предотвращение зависимости от незафиксированного обновления

10.4.4. Уровни изоляции в SQL

Подходы к разрешению проблем параллелизма в SQL претерпевали изменения по мере развития языка. Первоначально стандарт предписывал такую организацию одновременного выполнения операторов, чтобы это соответствовало ситуации, когда ни один из операторов не вводится до полного завершения предыдущего. В этой ситуации в любой момент времени только одна транзакция может иметь возможность изменять данные. Однако при большом числе одновременных корректирующих обращений такой подход может значительно снизить производительность системы. Поэтому (в тех случаях, когда это возможно, ищется компромисс между обеспечением строгой целостности базы данных в любой момент времени и производительностью.

В SQL-92 определены так называемые уровни изоляции (isolation level).

1. Уровень SERIALIZABLE (последовательное выполнение) -обеспечивает максимальную степень целостности и соответствует требованиям предыдущих стандартов. Обычно именно этот уровень устанавливается по умолча-

нию. При его выборе каждая транзакция выполняется изолированно.

2. Уровень REPEATABLE READ (повторяющееся чтение) - допускает вставку новой записи в таблицу, обрабатываемую транзакцией. При этом в принципе может возникать эффект так называемой фантомной вставки.

3. Уровень READ COMMITTED (чтение с фиксацией) - допускает выполнение запроса при условии, что результаты параллельных транзакций были зафиксированы.

4. Уровень READ UNCOMMITTED (чтение без фиксации) - допускает выполнение запроса независимо от того, были зафиксированы результаты параллельных транзакций или нет.

Если транзакция объявлена как READ ONLY, то это переводит ее на уровень READ UNCOMMITTED.

Уровень изоляции определяется в предложении SET TRANSACTION, которое имеет следующий синтаксис:

SET TRANSACTION {ISOLATION LEVEL
{READ UNCOMMITTED
| READ COMMITTED
| REPEATABLE READ
| SERIALIZABLE}
| {READ ONLY | READ WRITE}
| {DIAGNOSTICS SIZE число условий}}.,..;

Предложение DIAGNOSTICS SIZE определяет количество элементов, используемых для сохранения диагностической информации.

Уровень изоляции и характер транзакции (только чтение или чтение/запись) являются взаимозависимыми. Если задано READ WRITE, то ISOLATION LEVEL не может быть READ UNCOMMITTED. Если ISOLATION LEVEL определен как READ UNCOMMITTED, транзакция становится по умолчанию READ ONLY. В противном случае транзакция по умолчанию считается READ WRITE.

10.4.5. Использование хранимых процедур и триггеров для контроля целостности БД

Как отмечалось выше, при использовании технологии «клиент-сервер» запрос передается для выполнения на сервер. На практике часто встречается ситуация, когда один и тот же запрос выполняется многократно. Для сокращения времени выполнения таких запросов можно использовать хранимые процедуры. При этом запрос в оттранслированном виде хранится на сервере и при возникновении потребности в обработке данных на сервер передается не весь текст запроса целиком, а только обращение к соответствующей процедуре. Время выполнения запроса уменьшается не только за счет уменьшения сетевого трафика, но и в связи с тем, что процедура уже заранее и одно-

кратно транслирована, а оптимальный план ее выполнения можно определять сравнительно редко.

Другой способ выполнения некоторых стандартных для данной прикладной области операций - использование триггеров. Триггеры срабатывают каждый раз, когда выполняется заданная операция над заданной таблицей. В результате может быть выполнен один оператор SQL или вызвана хранимая процедура.

Хранимые процедуры и триггеры могут использоваться в разных ситуациях, в том числе и для контроля целостности БД.

10.5. Тиражирование данных

10.5.1. Основные понятия

Тиражирование - используемая в РБнД технология, предусматривающая поддержку копий всей БД или ее фрагментов в нескольких узлах сети. Копия базы данных, являющаяся членом набора других копий, которые могут быть синхронизированы между собой, называется *репликой*. Копии БД обычно приближены к местам использования информации. Как синоним понятию «тиражирование» используется термин «репликация». Тиражирование является сравнительно новой технологией.

Процесс обновления реплик, при котором происходит передача обновляемых записей и других объектов и согласование дублирующихся данных, называется *синхронизацией*. Обмен данными между репликами может быть как односторонним, так и двусторонним. Кроме того, возможна синхронизация реплик под управлением синхронизатора. В отличие от собственно распределенных систем (систем с фрагментированием), в которых, как правило, при выполнении распределенных запросов реализуется протокол двухфазной фиксации, в системах с реплицированными базами данных обычно используется инструментарий асинхронной репликации.

В настоящее время многие известные СУБД предлагают пользователям возможности репликации. Но, как и во всякой новой области, терминология и подходы к реализации отличаются от системы к системе.

В некоторых системах используются метафоры из издательской деятельности (издатель, публикация, подписчик).

Совокупность данных, которые могут подвергаться тиражированию, называется *публикацией*.

В системах с тиражированием присутствуют все функции, присущие другим видам распределенных систем, плюс еще специфические функции, вызванные именно тиражированием. Это функции, обеспечивающие пересылку изменений всем узлам-пользователям; функции поддержания идентичности всех копий (реплик) БД; если эталонная база - единственная, то функции формирования базы данных-эталона и некоторые другие. Причем часть этих функций может быть совмещена на одном узле, а часть - отсутствовать, в зависимости от использованной технологии тиражирования.

10.5.2. Преимущества и недостатки тиражирования

Преимущества. Использование технологии тиражирования имеет следующие преимущества:

- сокращение сетевого трафика при выполнении запросов;
- повышение доступности данных. Доступ к локальной копии БД обеспечивается даже в случае, если доступ к центральному серверу по той или иной причине невозможен;
- повышение производительности (за счет приближения данных к месту их использования, а также за счет специфики технологии выполнения запроса: не нужно ждать, чтобы все части распределенной БД были одновременно доступны);
- повышение автономности рабочих мест пользователей;
- повышение надежности системы (наличие множества копий повышает вероятность восстановления системы в критических ситуациях);
- уменьшение трафика (при определенных условиях);
- уменьшение конкуренции за ресурсы со стороны пользователей.

Недостатки. Дублирование данных при использовании технологии тиражирования влечет за собой следующие очевидные недостатки:

- дополнительный расход памяти;
- сложность обеспечения целостности данных; возможность возникновения конфликтов при корректировке;
- наличие временного лага между фиксацией события в БД и доступностью этой информации для всех пользователей сети;
- повышенные требования к рабочим станциям;
- системы с поддержкой тиражирования данных требуют тщательного продумывания схемы тиражирования. Само по себе проектирование современной БД представляет собой непростую задачу, а включение в нее схемы тиражирования данных требует дополнительного времени на проектирование и организацию.

10.5.3. Виды тиражирования

Существуют разные виды тиражирования. Они различаются степенью транзакционной целостности и степенью автономности узлов (что, как правило, находится в обратной зависимости друг от друга). Требования к механизму тиражирования зависят от задач, которые решает вся вычислительная система в целом.

Основные проблемы, возникающие в системах с тиражированием, связаны с поддержкой согласованности состояния всех копий. Существует множество схем проведения обновления копий в РБД.

По моменту внесения изменений в реплики различают синхронное и асинхронное тиражирование.

Асинхронное тиражирование (называемое также тиражированием с промежуточным хранением) реализует схему, при которой обновление всех копий баз данных может проводиться неодновременно. При таком методе поддержки логической целостности распределенной базы данных имеет место некоторая рассинхронизация состояния локальных баз данных во времени, т.е. изменение состояния одной локальной базы данных отстает от изменения другой локальной базы данных по времени.

Асинхронность обмена данными в целом ряде случаев вполне допустима, и при этом удается обойти, в частности, проблему ограниченной пропускной способности и недостаточной надежности сети. Если один сервер системы, требующий обновления тиражируемых данных, выходит из строя, то система продолжает работать с остальными, при этом обновление данных на сбойном сервере после восстановления его работоспособности произойдет автоматически, т.е. ошибка на одном узле глобальной сети не повлияет на работу остальных узлов.

Механизм асинхронного тиражирования данных особенно актуален при работе в глобальной сети с недостаточно надежными и быстродействующими каналами связи. Например, при использовании телефонных линий связи механизм двухфазной фиксации транзакции практически нереализуем, поскольку все оборудование одновременно в работоспособном состоянии почти никогда не находится.

Асинхронное тиражирование может быть периодическим или апериодическим. *Периодическое тиражирование* выполняется через заданные интервалы времени. Время выполнения очередных циклов тиражирования выбирается исходя из частоты возникновения изменений и их интенсивности, ограничения на допустимое время рае-синхронизации состояний источника и приемника, объема данных, передаваемых в единичном цикле тиражирования, и максимально допустимой загрузки коммуникационных ресурсов и некоторых других факторов.

Момент выполнения *апериодического тиражирования* определяется обычно каким-либо событием (такой механизм называется также тиражированием по событиям) и может быть реализован, например, через механизм процедур или триггеров.

В случае *синхронного тиражирования* предполагается завершение транзакции только после успешной модификации всех копий. Синхронное тиражирование использует механизм двухфазной фиксации (2PC) - Two-Phase Commit: основная система связывается с подчиненными копиями баз данных и одновременно вносит в них изменения, блокируя соответствующие записи. Если хотя бы одна из таких копий недоступна, изменения не выполняются. Поскольку механизм двухфазной фиксации предъявляет высокие требования к системе и снижает степень автономности узлов, то он в технологии тиражирования используется не часто.

Чтобы как-то компенсировать недостатки синхронного тиражирования, используют *тиражирование во времени, близком к реальному*, которое заключается в том, что сразу после завершения транзакции на основном узле

изменения автоматически передаются тиражируемым копиям. Но при данном виде тиражирования в случае недоступности одного из узлов изменения хранятся до восстановления связи с получателем. Такая схема создает меньшую загрузку сети и подходит для приложений, не требующих строгой синхронности копий данных.

Наиболее жестким критерием согласованности является критерий *полной эквивалентности копий*, который требует, чтобы по завершении транзакции все копии логического элемента данных были идентичны. Такая технология синхронизации не позволяет достичь одной из основных целей применения технологии тиражирования - повышения производительности за счет обеспечения большей автономности отдельных частей системы - и поэтому используется в системах с тиражированием редко.

Типичный протокол тиражирования, обеспечивающий сериализуемость по критерию полной эквивалентности копий, известен под названием Read-Once/Write-All (ROWA) - одно чтение, запись во все копии. Операция чтения в этом протоколе сводится к чтению одной какой-либо копии (вопрос о том, из какой именно копии будет выполняться чтение, может решаться из соображений эффективности). Каждая операция записи в логический элемент данных отображается на множество операций записи в физические копии этих элементов.

Протокол ROWA прост, но он требует доступности всех копий элемента данных, для того чтобы транзакция была завершена. Сбой на одном из узлов приведет к дублированию транзакции, что снижает доступность данных.

Было предложено несколько альтернативных алгоритмов, смягчающих требования ROWA относительно одновременности внесения изменения во все копии.

Реплики в РБД с тиражированием могут быть *равноправными* и *неравноправными*. Чаще всего они являются неравноправными. В этом случае одна из реплик считается основной. Даже При наличии неравноправных реплик возможны ситуации, при которых корректировать разрешается любую реплику, но часто изменения позволено вносить только в основную реплику, а другие реплики доступны пользователям только по чтению. Такая схема тиражирования называется *тиражированием из основного узла* (primary site). При этом данные асинхронно копируются из основной в иные реплики.

Поскольку при такой схеме тиражирования предъявляются повышенные требования к сохранности основной реплики и надежности функционирования основного узла, то иногда вышеописанную модель дополняют *горячим резервированием (*failover) основного узла. В этом случае основной узел тиражирует изменения как на подчиненные узлы, так и на узел с резервной копией. Если основной узел выходит из строя, то владельцем данных становится резервный, который с этого момента выполняет все его функции.

Одной из технологий, которые могут использоваться, если изменения вносятся в локальные реплики, является *тиражирование слиянием* (merge replication). Суть метода заключается в том, что операции выполняются на удаленном компьютере, который может быть даже полностью отключен от

компьютерной сети. Автономная СУБД записывает все операции с данными и их очередность. Затем в определенный момент автономный компьютер связывается с издателем (сервером, на котором корректируется БД, подлежащая дальнейшему тиражированию) и согласовывает с ним свои данные, пересылая издателю последовательность операций, проведенных в удаленной базе данных. При возникновении конфликтов они разрешаются с помощью различных алгоритмов.

В некоторых системах функции формирования «эталонной» базы и рассылки реплик конечным пользователям (подписчикам) распределены между разными узлами системы. В этом случае после слияния издатель передает изменения на сервер-дистрибьютер для дальнейшего распространения по подписчикам.

Тиражирование слиянием обеспечивает максимальную автономность удаленной базы данных.

При использовании этого, метода могут возникнуть коллизии, которые необходимо себе хорошо представлять и предотвращать их разнообразными средствами, как техническими, так и организационными.

Кроме того, существуют так называемые *поточные модели тиражирования* (workflow). В этой модели владелец данных меняется динамически: данные могут обновляться на различных узлах, но в любой момент времени такими полномочиями обладает только один из них. Узлы последовательно обновляют принадлежащие им данные и тиражируют изменения на всю систему.

Некоторые системы позволяют любым узлам с тиражируемыми копиями одновременно обновлять их. Такая модель называется моделью с *произвольным обновлением данных*. В этом случае возможно возникновение конфликтов между отдельными репликами, и необходим развитый механизм выявления и разрешения этих конфликтов, предотвращающий потерю целостности данных. При этом могут использоваться разнообразные алгоритмы разрешения возникающих коллизий. В некоторых системах реализован какой-либо один из возможных алгоритмов, в других - имеется возможность выбора из нескольких вариантов.

Еще одним критерием классификации механизмов тиражирования является признак того, кто является инициатором обновления реплик. Внесение изменений в реплики может инициироваться рабочими станциями. Такой процесс обновления называется *обновлением по запросу*. Обновление может выполняться как *по определенному графику*, так и *вручную*.

Другой подход заключается в том, что инициатором проведения изменений является сервер (*принудительная рассылка тиража*). Такая рассылка может осуществляться либо в момент появления изменений (*немедленное тиражирование*), либо по *графику* (через заданный интервал, в указанное время).

Немедленное тиражирование (называемое также *тиражированием по событиям*) означает, что сразу после завершения транзакции на основном узле изменения автоматически передаются тиражируемым копиям.

Метод, при котором считывание данных проводится удаленными узлами из основных, называется *вытягиванием* изменений, а метод, заключающийся в выгрузке данных на узлы с копиями,- выталкиванием изменений.

Обновление содержания реплик может быть обеспечено следующими способами:

- копированием моментального снимка базы данных;
- копированием и выполнением очереди подтвержденных издателем транзакций;
- копированием изменений из журнала БД.

Моментальный снимок базы данных (snapshot) отражает состояние базы данных в целом или ее фрагмента на момент получения снимка. При копировании моментального снимка базы данных часто передаются не только собственно данные, но и дополнительная служебная информация, например идентификаторы столбцов и строк.

В процессе *тиражирования транзакций* (transactional replication) от издателя к подписчикам передаются не данные, а операции над ними. Само обновление происходит на рабочей станции. Передачу транзакций можно использовать только в том случае, когда в автономной базе данных уже хранится копия основной БД. При использовании этой схемы периодически необходимо выполнять и полную синхронизацию данных, которая выполняется по методу тиражирования моментального снимка данных. Тиражирование транзакций лучше всего использовать, когда объем базы данных велик, а операций над ними выполняется немного.

В БнД вообще, а в распределенных - особенно одной из важнейших задач является обеспечение целостности баз данных, в том числе восстановление баз после их разрушения. Для этих целей используются *журналы изменений баз данных* - протокол, в котором регистрируются в хронологическом порядке исходные и обновленные состояния всех записей базы данных, модифицированных в процессе исполнения транзакций. Ведение журнала обеспечивается многими СУБД и применяется при процедурах восстановления.

В случае *обновления на подписчике* (immediate updating subscribers) тиражирование инициируется издателем. Как только издатель подтверждает транзакцию, он сообщает дистрибьютеру о том, что данные изменены. Дистрибьютер забирает подтвержденную транзакцию и рассылает ее подписчикам. Если во время завершения транзакции связь между дистрибьютером и издателем была прервана, то транзакция записывается в очередь и будет передана дистрибьютеру, как только связь восстановится. Дальнейшее распространение данных выполняет дистрибьютер либо принудительно, либо по запросу. Транзакции проводятся только через издателя.

10.6. Обеспечение целостности и безопасности данных в РБД

10.6.1. Особенности обеспечения целостности в РБД

Существует ряд проблем обеспечения ограничений целостности распределенной БД, помимо тех аспектов, которые присущи любым БД:

1. возможность одновременного доступа нескольких пользователей к одной и той же информации (особенно если эти обращения к БД - корректирующие);
2. физический разброс отдельных частей БД по разным компьютерам;
3. разнотипность источников информации.

Первая проблема имеет место в любых РБнД, вторая - если база данных является распределенной, третья - если система является гетерогенной.

Первая группа проблем в области обеспечения целостности в распределенных БнД обусловлена в основном возникновением опасности искажения данных при их одновременной корректировке разными пользователями. В зависимости от способа организации БД проблема приобретает специфические черты, обусловленные используемой технологией обработки данных. Так, проблема обеспечения целостности данных будет по-разному решаться при корректировке единой централизованной БД, при работе с несколькими разными копиями БД (репликами) или при работе с распределенной базой данных, отдельные части которой не дублируют друг друга.

Возможны разные схемы обеспечения целостности данных при выполнении корректирующих обращений в многопользовательском режиме:

- запрещение корректировки информации, если ее корректирует другой пользователь (*блокировка*);
- *корректировка* разных копий информационных единиц и последующее устранение возникающих коллизий.

Если СУБД предоставляет возможность выбора способа обеспечения целостности при многопользовательских обращениях, то на результат этого выбора будут влиять многие факторы, в том числе:

- степень конкуренции при выполнении корректирующих обращений - насколько часто возникает ситуация одновременной корректировки одной и той же информационной единицы;
- ограничения на время реакции системы,
- требования к актуальности и непротиворечивости данных в каждый момент времени;
- характеристика технических средств.

Вторая группа проблем обеспечения целостности в распределенных системах вызвана распределением данных и, как следствие, распределением процедур их обработки, т.е. это проблемы, обусловленные именно разнесением данных на разные узлы системы.

Как известно, существует два подхода к обеспечению целостности в распределенных информационных системах - *строгая целостность* (tight consistency) и *нестрогая целостность* (loose consistency). Первый вариант га-

рантирует целостность данных в любой момент времени, например, с помощью двухфазного протокола фиксаций (2PC). Обеспечение строгой целостности требует высокого качества коммуникаций, поскольку все узлы должны быть постоянно доступны. Второй подход допускает наличие временной задержки между внесением изменений в публикуемую базу и их отражением на узлах подписчиков.

Протокол двухфазной фиксации транзакции состоит в последовательном прохождении базы данных в процессе выполнения транзакции через два этапа. Первый этап (первая фаза) - выполнение синхронизированного захвата всех объектов данных, к которым имело место обращение от имени транзакции. Объекты данных захватываются на всех серверах. На втором этапе (во второй фазе) либо происходят все изменения на всех серверах, либо, в случае хотя бы одной ошибки, происходит откат к состоянию, в котором находилась база данных до выполнения первого этапа.

Механизм двухфазной фиксации транзакции имеет ряд недостатков:

- захват всех необходимых данных на всех серверах может надолго заблокировать доступ к данным;
- велика вероятность отказа от обновления из-за какой-нибудь, пусть единичной, ошибки;
- если какой-либо сервер или выход в глобальную сеть окажется недоступным, произойдет потеря транзакции;
- использование в структуре сети координирующего узла связано с дополнительной опасностью, поскольку выход его из строя приведет к блокировке данных, затронутых транзакцией, до тех пор, пока он не будет восстановлен;
- сложность обработки транзакции при использовании этого протокола сама служит источником дополнительного трафика, что увеличивает время реакции системы.

Кроме того, недостаточная пропускная способность сети и малая скорость передачи данных могут увеличить время реакции до недопустимого уровня.

Разные СУБД поддерживают разные технологии обеспечения целостности.

10.6.2. Средства защиты данных

Способы защиты данных

При работе в многопользовательском режиме особую актуальность приобретает защита данных от несанкционированного доступа. Существуют различные приемы управления доступом к базе данных. Эти приемы обеспечивают разный уровень безопасности. Некоторые из них присущи любым информационным системам, другие - конкретным СУБД.

Шифрование/дешифрование. *Шифрование* базы данных - это простейший способ защиты, при котором ее файл видоизменяется и становится недоступным для чтения с помощью стандартных служебных программ или текстовых редакторов. Шифрование незащищенной базы данных неэффективно, поскольку можно непосредственно открыть исходную базу данных и получить полный доступ к ее объектам. Шифрование обычно применяется при электронной передаче базы данных или сохранении ее на внешний носитель (дискету, компакт-диск и т.п.)

Дешифрование базы данных - это операция, обратная шифрованию.

Отображение и скрытие объектов. Другим способом защиты объектов в базе данных от посторонних пользователей является скрытие всей базы данных при просмотре каталогов средствами операционной системы или скрытие отдельных объектов БД при работе с базой данных средствами конкретной СУБД. Этот способ защиты не является достаточно надежным, поскольку скрытые объекты относительно просто можно отобразить.

Использование параметров запуска. Эти параметры позволяют задать стартовую форму, которая автоматически открывается при открытии базы данных. При определении формы можно скрыть окно базы данных, предоставляемое СУБД, и установить собственную кнопочную форму. При этом пользователь может выполнять с базой данных только те действия, которые допускает интерфейс, предоставляемый данным приложением.

Использование пароля. Другим простейшим способом защиты является установка пароля для открытия базы данных. После установки пароля при каждом открытии базы данных будет появляться диалоговое окно, в которое требуется ввести пароль. Только те пользователи, которые введут правильный пароль, смогут открыть базу данных.

В принципе может быть установлен единый пароль для всех пользователей БД. Но наиболее гибким и распространенным способом защиты базы данных является защита на уровне пользователя, при котором каждому пользователю присваивается пароль. При запуске СУБД каждый пользователь должен быть идентифицирован; система проверяет соответствие идентификатора пользователя и его пароля.

Для каждого пользователя могут быть определены не только уникальный код, но и уровень доступа, и объекты, доступ к которым получает пользователь.

Многие СУБД позволяют кроме единичных пользователей создавать еще и их группы.

Запрещение репликации базы данных. Как указывалось выше, репликация позволяет пользователям создавать копию общей базы данных. Это может быть в принципе использовано и для дальнейшего нелегального распространения реплицированных данных. Поэтому в некоторых случаях может потребоваться запретить репликацию базы данных.

Запрещение установки паролей и настройки параметров запуска пользователями. При многопользовательском доступе к данным важно обеспечить не только сохранность и конфиденциальность данных, но и воз-

можность доступа к данным всем тем пользователям данных, для которых это необходимо. Поэтому СУБД должна иметь механизмы, не позволяющие любым пользователям устанавливать пароль на БД, поскольку, если это произойдет, никто, не зная пароля, не сможет открыть базу данных.

Также желательно иметь механизм установки запрета на изменение параметров запуска, которые определяют такие свойства, как настраиваемые меню, настраиваемые панели инструментов и стартовую форму.

Создание и удаление пользователей

При работе в многопользовательской среде большое значение приобретает понятие *пользователь базы данных* - владелец определенного набора объектов базы данных.

Пользователи системы могут быть разделены на классы. В системе любого размера всегда имеются некоторые типы суперпользователей - пользователей, которые автоматически имеют большинство (или все) привилегий и могут передать свой статус суперпользователя кому-нибудь с помощью привилегии или группы привилегий. *Администратор базы данных* (DBA) является термином, наиболее часто используемым для такого суперпользователя и для привилегий, которыми он обладает.

Других пользователей создают Администраторы баз данных; они же дают им начальные привилегии. Создавать пользователей могут только администраторы. Давать права пользователям и отбирать их могут не только администраторы, но и другие пользователи, обладающие соответствующими правами.

Пользователи могут объединяться в группы. *Группа пользователей* - это пользователи, наделенные одинаковым набором привилегий. Один и тот же пользователь в принципе может входить в разные группы. Каждый пользователь имеет специальное идентификационное имя или номер (Authorization ID).

Поскольку большинство промышленно эксплуатируемых корпоративных СУБД являются SQL-серверами, рассмотрим вопросы управления пользователями на примере SQL-систем.

Конкретные формы процесса управления пользователями в различных СУБД могут значительно отличаться друг от друга. Процесс управления пользователями в большой мере зависит от используемой операционной системы, архитектуры БД. В связи с этим в соответствующей части SQL наблюдается большая зависимость от платформы и производителя. До SQL-92 в стандарте не затрагивалось большинство возникающих в этой сфере вопросов. В SQL-92 часть вопросов нашла свое отражение, но требования их реализации не являются обязательными.

Большинство СУБД содержат высокоуровневые инструменты администрирования, обеспечивающие управление пользователями, но в каждой системе это делается по-своему, что осложняет не только процесс стандартизации, но и представление этих вопросов в общем виде в учебнике.

Процесс управления пользователями можно разбить на три главных этапа. Сначала необходимо создать учетную запись пользователя в базе данных. Далее пользователя необходимо наделить привилегиями сообразно тем задачам, которые пользователь предположительно будет решать в рамках базы данных. Наконец, после того, как доступ к данным пользователю будет уже не нужен, необходимо либо удалить из базы данных его учетную запись, либо отменить ранее предоставленные ему привилегии.

Перед началом работы с БД пользователь должен быть идентифицирован с помощью процедуры входа, обычно включающей запрос имени и пароля пользователя. После входа запускается сеанс (sessions) работы с СУБД.

Определение и отмена привилегий

Распределенные БнД предполагают работу с базой данных многих пользователей. Однако не всем пользователям следует разрешать выполнять любые действия с базой данных. Поэтому пользователям предоставляются привилегии. *Привилегия* - это право пользователей на выполнение определенных операций над объектами данных некоторого типа.

Привилегии в разных литературных источниках классифицируются поразному. В [10] привилегии баз данных делятся на две категории: *системные привилегии* (system privileges) и *объектные привилегии* (object privileges). Системные привилегии контролируют общий доступ к базе данных. К ним относятся право создавать таблицы и другие объекты, а также право администрировать базу данных.

Объектные привилегии связаны с конкретным объектом базы данных. Объектная привилегия логически состоит из трех частей:

- объекта, к которому применяется привилегия;
- операции, которые она разрешает;
- пользователя, которому даются эти привилегии.

Одна из первых привилегий, которая должна быть определена, - это привилегия создателей таблиц. Если все пользователи будут иметь возможность создавать в системе базовые таблицы, это может привести к избыточности данных, их несогласованности и, как следствие, к неэффективности системы.

Пользователь, создавший таблицу, является ее владельцем. Это означает, что пользователь имеет все привилегии в созданной им таблице и может передавать привилегии другим пользователям в этой таблице.

Каждый пользователь в среде SQL имеет специальное идентификационное имя (или номер).

Привилегии даются оператором GRANT (ПРЕДОСТАВИТЬ) и отменяются оператором REVOKE (ОТМЕНИТЬ).

Оператор GRANT имеет следующий синтаксис:

GRANT привилегия.,..**ON** имя объекта
TO {пользователь, которому предоставляется привилегия.,..}|**PUBLIK**

[WITH GRANT OPTION];

привилегия:=
{ALL PRIVILEGES}
| {SELECT
| DELETE
| {**INSERT** [(имя столбца.,..)]}
| {**UPDATE** [(имя столбца.,..)]}
| {**REFERENCES** [(имя столбца.,..)]}
| **USAGE**}.

Когда SQL получает оператор GRANT, он проверяет привилегии пользователя, подавшего эту команду, чтобы определить, допустим ли оператор GRANT

Для пользователя таблицы могут быть назначены следующие типы привилегий:

- SELECT - разрешение выполнять запросы в таблице.
- INSERT - разрешение выполнять оператор INSERT (вставка новой строки) в таблице.
- UPDATE - разрешение выполнять оператор UPDATE (обновление значений полей) в таблице. Можно ограничить эту привилегию для определенных столбцов таблицы.
- DELETE - разрешение выполнять оператор DELETE (удаление записей) в таблице.
- REFERENCES - разрешение определить внешний ключ.

В одном операторе GRANT можно назначить несколько привилегий, перечислив их через запятую, или использовать аргумент ALL, означающий, что пользователю передаются все привилегии для данной таблицы.

В одном операторе GRANT можно назначить привилегии нескольким пользователям одновременно, перечислив их через запятую, или использовать аргумент PUBLIC, означающий, что привилегии передаются все пользователям. Однако последней возможностью нужно пользоваться с осторожностью, так как PUBLIC означает не только текущих пользователей, но и всех пользователей, которые могут быть введены в систему в дальнейшем.

Предположим, что пользователь Digo является владельцем таблицы «Sotrudnik» и хочет позволить пользователю Ivanov выполнять запросы к ней. В этом случае пользователь Digo должен ввести команду

GRANT SELECT ON Sotrudnik **TO** Ivanov;

Предложение WITH GRANT OPTION позволяет передать пользователю возможность назначать привилегии для этой таблицы. Если, например, команда выглядит следующим образом:

GRANT SELECT ON Sotrudnik
TO Ivanov **WITH GRANT OPTION**;

то пользователь Ivanov получает возможность, в свою очередь, передавать право назначать привилегии другим пользователям, т. е. пользователь Ivanov может задать следующую команду:

GRANT SELECT ON Digo.Sotrudnik
TO Petrov **WITH GRANT OPTION**;

Обратите внимание на то, что когда на таблицу ссылается пользователь, не являющийся владельцем схемы, то перед именем таблицы указывается имя схемы.

Большинство привилегий объекта использует один и тот же синтаксис. Из перечисленных выше привилегий исключение составляют UPDATE и REFERENCES.

При задании привилегии UPDATE можно использовать тот же синтаксис, который применялся выше. Это будет означать, что пользователю дается право обновлять содержимое всех столбцов таблицы. Можно также после названия привилегии в скобках указать имена столбцов (если их несколько, то имена указываются в любой последовательности через запятую), на которые распространяется данная привилегия. Например, привилегия UPDATE может выглядеть следующим образом:

GRANT UPDATE (dolgnost, oklad) **ON** Sotrudnik **TO** Ivanov;

При задании привилегии REFERENCES также задаются имена столбцов.

Чтобы ограничить возможность просмотра таблицы только отдельными столбцами, следует воспользоваться механизмом создания представлений и назначать привилегии не для реальной таблицы, а для представления. Представления могут использоваться также и для обеспечения возможности ограничить просмотр только определенными строками или даже только, например, итоговыми или иными производными данными, если в операторе CREATE VIEW соответствующим образом задан «внутренний» SELECT.

Отмена привилегий осуществляется с помощью оператора REVOKE. Эта команда имеет синтаксис, схожий с синтаксисом оператора GRANT. Например, отмена привилегии на просмотр таблицы «Sotrudnik» для пользователя Ivanov будет выглядеть следующим образом:

REVOKE SELECT ON Sotrudnik **TO** Ivanov;

В конкретных СУБД могут поддерживаться привилегии, отличающиеся от привилегий, приведенных выше. Так, в некоторых СУБД имеется возможность задавать привилегию INDEX, которая позволяет пользователям созда-

вать индексы. Но объект INDEX в стандарте SQL не определен, и синтаксис команды задания этой привилегии может отличаться от системы к системе.

Стандартом SQL не определено, кто имеет право отменять привилегии. Однако обычно действует подход, при котором привилегии отменяются тем пользователем, который их предоставил, и отмена будет «каскадироваться», т.е. она будет автоматически распространяться на всех пользователей, получивших эту привилегию от пользователя, для которого привилегия отменена.

10.7. Работа в распределенной среде при использовании СУБД Access

10.7.1. Способы совместного использования данных в Access

Access, как известно, относится к классу настольных СУБД. Естественно, что сетевые возможности наиболее развиты в корпоративных СУБД, но в связи с повсеместной распространенностью сетевых технологий настольные СУБД также обеспечивают работу в многопользовательском режиме.

Access, как и большинство других СУБД, тоже обеспечивает возможность многопользовательской работы с базой данных. Допускается до 255 одновременно работающих с базой данных пользователей.

Кроме самостоятельного использования Access может применяться как средство проектирования при создании РБнД, а также в качестве интерфейсного средства при работе в гетерогенном РБнД; как правило, Access используется совместно с MS SQL-сервером.

Как указывалось выше, в Access используется своеобразное понятие базы данных: базы данных включают в своем составе не только таблицы и связи между ними (что традиционно считается базой данных), но и запросы, отчеты, модули, макросы и другие объекты. В связи с этим при работе в многопользовательском режиме возникают дополнительные проблемы и имеются специальные средства для их разрешения.

Существует несколько способов совместного использования данных в многопользовательской среде.

1. Совместное использование всей базы данных Microsoft Access.
2. Совместное использование таблиц базы данных Microsoft Access.
3. Совместное использование объектов базы данных Microsoft Access или страниц доступа к данным в Интернете.
4. Репликация базы данных Microsoft Access.
5. Создание приложения в архитектуре «клиент-сервер».

Различие в первых двух возможностях обусловлено особенностями архитектуры Access и является специфичным именно для этой СУБД. Напомним, что в Access база данных - это не только таблицы с данными, но и формы, отчеты, запросы, макросы и модули.

Можно поместить всю базу данных Access (т.е. базу данных, содержащую все эти объекты) на сетевой сервер или в общую папку. Далее следует убедиться, что для базы данных задано открытие в режиме совмест-

ного доступа. Этот режим используется по умолчанию. Чтобы установить этот режим, необходимо в меню **Сервис** выбрать команду **Параметры**, затем на вкладке **Другие** в группе параметров **Режим открытия по умолчанию** выбрать параметр **общий** доступ (рис. 10.8).

Путем соответствующей настройки можно также определить параметры, управляющие блокировкой данных и обновлением данных в общей базе данных, такие, как «блокировка по умолчанию» (определяет блокируемую информационную единицу), «число повторов обновления» (определяет число попыток обновления записи, если она оказалась заблокированной) и некоторые другие.

Совместное использование всей базы данных является наиболее простым способом организации совместного доступа. Все работают с одними и теми же данными, формами, отчетами, запросами, макросами и модулями. Это приемлемо, когда информационные потребности всех пользователей совпадают.

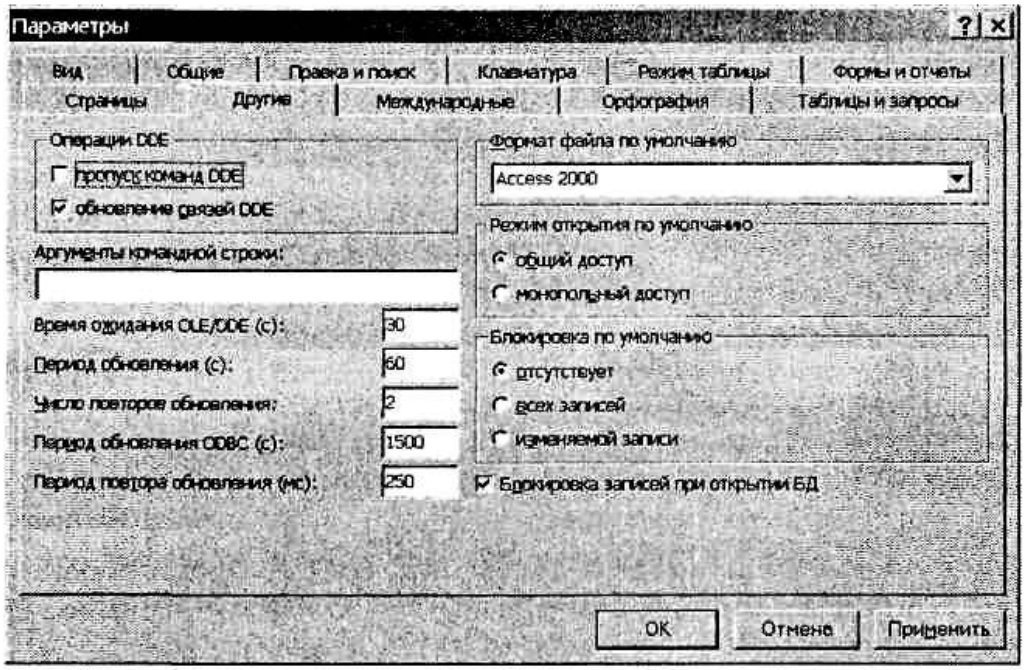


Рис. 10.8. Настройка параметров при совместном использовании БД

Существует и другой путь - разделение базы данных Microsoft Access на две части: собственно данные и остальные объекты базы данных Microsoft Access. В этом случае можно поместить на сетевой сервер только таблицы базы данных, а остальные объекты хранить на компьютерах пользователей. При этом работа с базой данных Access происходит быстрее, так как по сети передаются только данные. Если разделить базу данных на серверную часть (таблицы) и клиентскую часть, пользователь получит возможность изменять формы, отчеты и другие объекты в собственной клиентской базе данных, не влияя при этом на других пользователей.

Для того чтобы выполнить разделение баз данных, необходимо выполнить следующие шаги:

- в меню **Сервис** выбрать команду **Служебные программы** и подкоманду **Разделение баз данных**;
- в ответ на вопрос Мастера «Разделить базу данных» указать место на диске, где будет размещена отделенная БД, т.е. часть исходной базы данных, которая содержит таблицы.

При формировании запросов таблицы, к которым они относятся, могут находиться в другой БД. При этом используется так называемое присоединение таблиц. Администратор сети может создать несколько копий стандартной БД с присоединенными таблицами или позволить пользователю создавать собственные БД и присоединять таблицы самостоятельно. Пользователи могут корректировать данные в присоединенной таблице, но не могут менять ее структуру.

Преимущества использования присоединения:

- доступ к БД осуществляется быстрее, поскольку отсутствуют траты времени на передачу объектов по сети;
- резервирование данных и обновление других объектов упрощается, если таблицы хранятся отдельно от остальных объектов БД;
- пользователь может менять свои формы и отчеты, не оказывая влияние на компоненты, разработанные другими пользователями.

Присоединенные таблицы могут находиться не только в другой БД Access, но даже в БД других СУБД.

Распределенные базы данных могут быть расположены как в локальной, так и в глобальной сети. Обеспечить совместное использование данных из БД при работе в Интернете можно несколькими способами. Имеется возможность вывести один (или несколько) объект базы данных в формате статического HTML или генерируемого сервером HTML, а также создать страницы доступа к данным, а затем отображать их в обозревателе, например в Microsoft Internet Explorer, в Интернете.

Существуют различные способы выполнения репликации и синхронизации баз данных в Access:

портфельная репликация;

- команды репликации в меню Сервис Microsoft Access;
- репликация в проекте Microsoft Access;
- программная репликация;
- диспетчер репликации Microsoft.

При использовании нескольких компьютеров, например стоящего в офисе и переносного, имеется возможность создавать с помощью портфеля Microsoft Windows реплики базы данных Microsoft Access и поддерживать синхронизацию этих реплик. При этом находящиеся в разных местах пользователи могут одновременно работать каждый со своей копией, а затем синхронизировать их через подключение удаленного доступа или через Интернет.

При работе в среде «клиент-сервер» можно использовать имеющиеся дополнительные возможности и способы защиты, обеспечиваемые сервером. Данные сохраняются в таблицах на сервере базы данных, например на Microsoft SQL Server, а не в локальных таблицах в Microsoft Access. Приложение Access (клиент) принимает требующиеся ему данные с сервера. Сервер поддерживает целостность данных и выполняет любые запросы, которые он может распознать.

При открытии файла базы данных Access (.mdb) в режиме общего доступа Microsoft Access создает файл сведений о блокировке (.ldb) с тем же именем (например, Учебная.ldb) и в той же папке, что и файл базы данных. В файле сведений о блокировке сохраняются имя компьютера и имена всех совместно работающих пользователей базы данных в системе защиты. Microsoft Access использует эти сведения для управления совместной работой. В большинстве случаев Microsoft Access автоматически удаляет файл сведений о блокировке при закрытии файла базы данных последним пользователем.

В многопользовательской среде с одними и теми же записями могут одновременно работать несколько человек. Поскольку в то время, когда один пользователь пытается редактировать записи, другие пользователи также могут вносить в них изменения или даже удалять данные, при работе иногда возникают противоречия.

Microsoft Access позволяет отслеживать состояние записей во время внесения в них изменений и обеспечить использование самых свежих данных. Если два (или более) пользователя пытаются внести изменения в одну и ту же запись, на экране появляется сообщение, помогающее разрешить конфликт. Например, при попытке сохранить запись, заблокированную другим пользователем, на экране появляется имя заблокировавшего ее пользователя.

Чтобы облегчить отслеживание состояния записей, Microsoft Access отображает специальные маркеры в области выделения текущей записи, которые позволяют понять, в каком состоянии находится запись.

Одной из возможностей поддержки удаленных пользователей с помощью реплицирования в СУБД Access является использование команды **Репликация** в меню **Сервис** Microsoft Access. Для этого в меню **Сервис** следует выбрать позицию **Репликация**, а затем - **Создать реплику** (рис. 10.9). Реплицировать можно только закрытую базу данных. При этом она не должна быть защищена паролем.

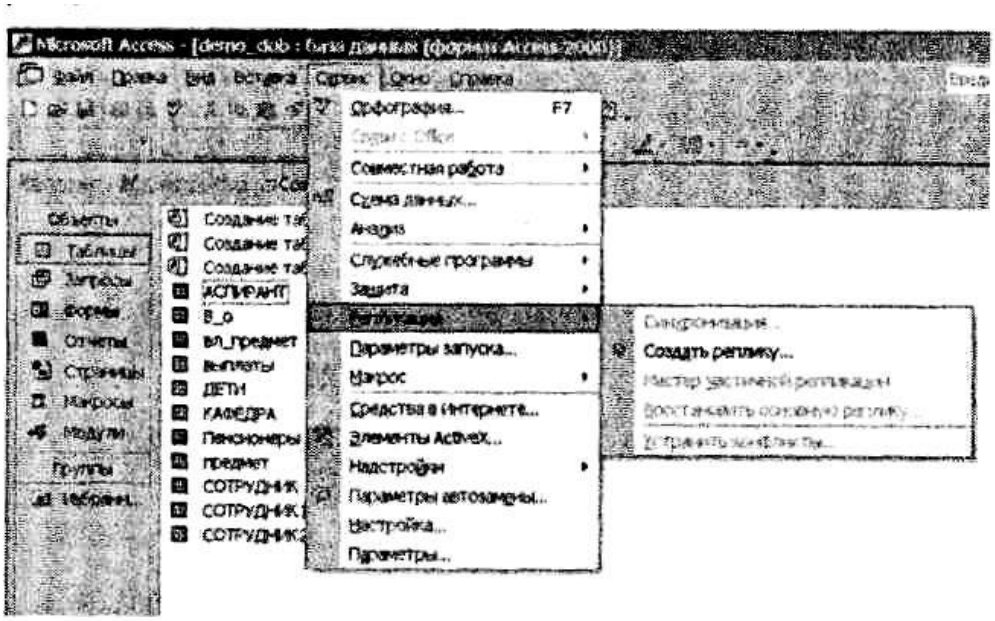


Рис. 10.9. Выбор команды **Репликация** в меню

Реплицирование может применяться не только при работе нескольких пользователей в сети, но и, например, при использовании как настольного, так и переносного компьютера. При этом на переносном компьютере создается «оторванная» реплика, с которой можно работать автономно. Впоследствии, когда появится возможность, в случае необходимости (если в реплику вносились изменения, которые должны быть учтены в основной базе данных) можно произвести синхронизацию.

Поскольку исходная БД при репликации изменяется, то Перед созданием реплики следует создать резервную копию исходной базы данных.

Реплицирование может использоваться не только при работе нескольких пользователей в сети, но и, например, при использовании как настольного, так и переносного компьютера.

В Microsoft Access 2000 и более поздних версиях реплике в наборе реплик при ее создании назначается приоритет, и в случае конфликта синхронизации применяется реплика с наивысшим приоритетом. При равенстве приоритетов используется реплика с минимальным значением свойства «ReplicaID».

10.7.2. Виды блокировок

Блокировкой называется запрещение некоторых операций над данными (чаще - корректировки информации), если их обрабатывает (корректирует) другой пользователь.

В Microsoft Access возможно выбрать одно из трех возможных значений параметра блокировки.

1.«*Отсутствует*». При выборе этого параметра изменяемая запись не блокируется. При попытке сохранить измененную запись, которую в это же время изменял другой пользователь, на экране появляется сообщение, пред-

лагающее следующие возможности: сохранить данный вариант записи, уничтожив изменения, внесенные другим пользователем; скопировать свою измененную запись в буфер обмена; отказаться от собственных изменений. Этот параметр позволяет изменять записи в любое время, но при внесении изменений разными пользователями могут возникать конфликты. Если вероятность возникновения конфликтов велика, а квалификация и ответственность пользователей - недостаточно высокая, то использование такого значения параметра блокировки опасно и может привести к нарушению целостности данных.

2.«*Изменяемой записи*». Microsoft Access блокирует изменяемую в данный момент запись, не позволяя изменять ее другим пользователям. При этом заблокированными могут оказаться другие записи, расположенные рядом на диске. Выбор данного значения параметра блокировки гарантирует завершение всех начатых изменений.

3.«*Всех записей*». Microsoft Access блокирует все записи формы или объекта в режиме таблицы (и базовых таблиц) на весь период их открытия, поэтому другие пользователи не могут изменить или заблокировать записи. Этот параметр накладывает слишком жесткие ограничения. Его следует использовать в случае выполнения массовой корректировки БД либо при выполнении какой-либо приоритетной задачи, предъявляющей жесткие требования ко времени ее выполнения.

При изменении данных из связанной таблицы базы данных SQL, поддерживающей протокол ODBC (Open DataBase Connectivity - стандарт интероперабельной работы с данными), Microsoft Access не блокирует записи. Вместо этого действуют правила блокировки этой базы данных SQL. В этом случае, независимо от выбранного параметра блокировки для конкретной базы данных, Microsoft Access ведет себя так, как будто выбрано значение «Отсутствует».

Кроме того, в Microsoft Access возможна временная ручная блокировка поля. Чтобы заблокировать поле, запретив изменение текущего значения поля, необходимо выделить это поле и нажать клавиши [Ctrl]+[F11]. Чтобы снять блокировку и разрешить обновление значения поля, следует выделить поле и нажать клавиши [Ctrl]+[Shift]+[F11].

10.7.3. Проекты Microsoft Access

Проект Microsoft Access является файлом данных Microsoft Access, имеющим расширение (.adp). Проект Microsoft Access обеспечивает эффективный доступ к базам данных Microsoft SQL Server с помощью архитектуры компонентов OLE DB. Используя проект Microsoft Access, можно легко создавать приложения в архитектуре «клиент-сервер». Такое приложение в архитектуре «клиент-сервер» может быть традиционным решением, основанным на формах и отчетах, или решением на базе Интернета, использующим страницы доступа к данным, либо сочетать в себе весь спектр возможностей. Проект Microsoft Access можно подключить к удаленной базе данных

Microsoft SQL Server, локальной базе данных Microsoft SQL Server или к локальной установке Microsoft SQL Server 2000 Desktop Engine (MSDE).

Проект Microsoft Access содержит только программные или HTML-объекты базы данных: формы, отчеты, имена и местоположение страниц доступа к данным, макросы и модули. В отличие от базы данных Microsoft Access, проект Microsoft Access не содержит объекты, основанные на данных или на определениях данных. Такие объекты, как таблицы, представления, схемы базы данных, сохраненные процедуры или определяемые пользователем функции, хранятся в базе данных Microsoft SQL Server.

Для создания приложения и доступа к данным проект Microsoft Access следует подключить к базе данных Microsoft SQL Server. Для подключения необходимо в меню **Файл** выбрать команду **Подключение**, открывающую диалоговое окно **Свойства связи с данными**, или использовать Мастер баз данных.

Работа с проектом Microsoft Access очень похожа на работу с базой данных Microsoft Access. Процесс создания форм, отчетов, страниц доступа к данным, макросов и модулей практически не отличается от используемого при создании базы данных Microsoft Access.

Подключившись к базе данных Microsoft SQL Server, можно просматривать, создавать, изменять и удалять таблицы, представления, сохраненные процедуры, определяемые пользователем функции и схемы баз данных с помощью конструктора баз данных, конструктора таблиц, конструктора запросов, построителя запросов и текстового редактора SQL. В проекте Microsoft Access содержится много Мастеров, таких же, как в базе данных Microsoft Access, например Мастер форм, Мастер отчетов, Мастер страниц и Мастер масок ввода. Эти Мастера помогают пользователю быстро создать прототипы или простые приложения, а также упрощают создание сложных приложений.

При создании нового проекта имеются две возможности: Проект (новые данные) (рис. 10.10)и Проект (существующие данные) (рис. 10.11). И в том, и в другом случае необходимо указать сервер, на котором будут храниться данные. Если используется существующая база данных, то следует выбрать именно ее.

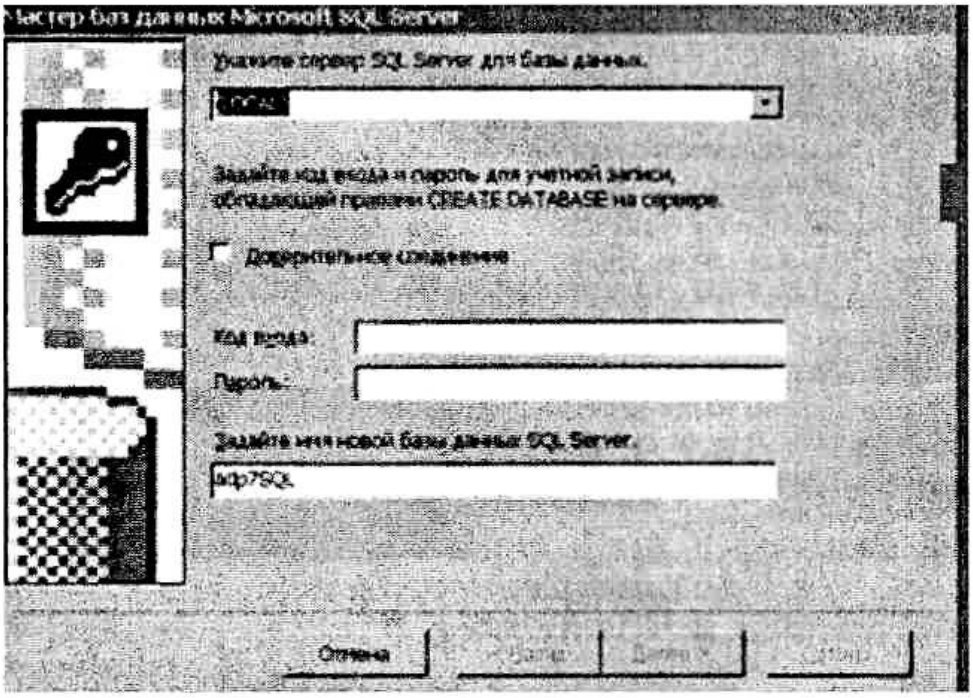


Рис. 10.10. Вид начального окна создания проекта (новые данные)

Имеется возможность подключить проект Microsoft Access к базе данных Microsoft SQL Server.

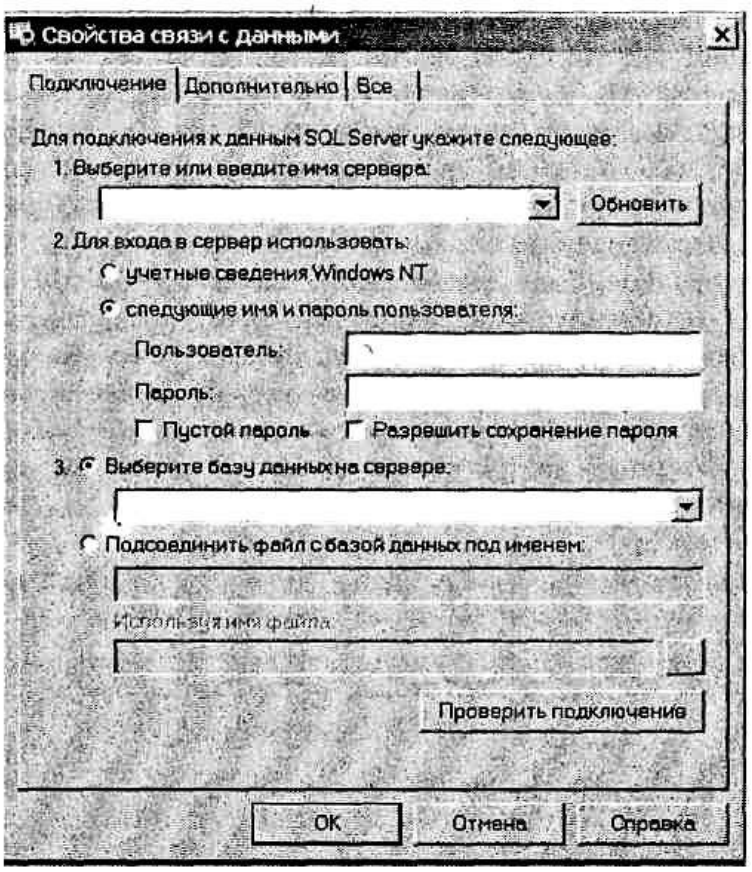


Рис. 10.11. Вид начального окна создания проекта (существующие данные)

10.7.4. Средства защиты Microsoft Access

Управление правами доступа пользователей

Защита на уровне пользователей Microsoft Access напоминает механизмы безопасности, используемые в корпоративных системах. Администратор базы данных может задать для каждого пользователя пароль и допустимые для него права (или, как они названы в Microsoft Access, - разрешения). Каждый пользователь представлен в системе своей учетной записью.

Ряд пользователей могут иметь одинаковый набор прав. Для того чтобы не задавать права для каждого из них, можно создавать так называемые рабочие группы. Рабочая группа - это группа пользователей, которые совместно используют данные в многопользовательской среде. Для всех пользователей, входящих в состав одной и той же группы, задаются одни и те же права.

В Microsoft Access определены две стандартные группы: администраторы (группа Admins) и пользователи (группа Users), но допускается определение дополнительных групп.

Существуют два типа разрешений на доступ: явные и неявные. *Явные* разрешения присваиваются непосредственно учетной записи пользователя. *Неявные* разрешения присваиваются учетной записи группы. Все пользователи, включенные в группу, получают все разрешения, предоставленные группе; удаление пользователя из группы лишает его всех разрешений, присвоенных данной группе.

Управление защитой на уровне пользователей для большинства баз данных является достаточно сложной задачей. Для упрощения ее выполнения в Microsoft Access имеется Мастер защиты.

Мастер защиты запускается из меню **Сервис/Защита/Мастер** (рис. 10.12). Для запуска Мастера база данных должна быть открыта в режиме общего доступа. Мастер защиты создает незащищенную копию базы данных, а затем защищает БД.

По умолчанию защищаются все объекты БД, а также те объекты, которые будут созданы после использования Мастера.

Мастер защиты позволяет создать группы пользователей и отдельных пользователей, включать пользователей в группы (рис. 10.13).

Как видно из рис. 10.13 существуют разные типы групп, для которых разрешен определенный уровень прав. Так, «Операторы архива» могут только открывать БД с монопольным доступом для резервирования и сжатия, но не могут видеть объекты БД. Члены группы «Все права на данные» могут изменять данные, но не могут менять макет ни одного объекта БД. Члены группы «Полные права» имеют все разрешения на все объекты, но не могут назначать разрешения для других пользователей. Члены группы «Разработчики проекта» имеют разрешения на изменение данных и всех объектов, но не могут изменять таблицы и связи. Разрешения для других групп ясны из их названия.

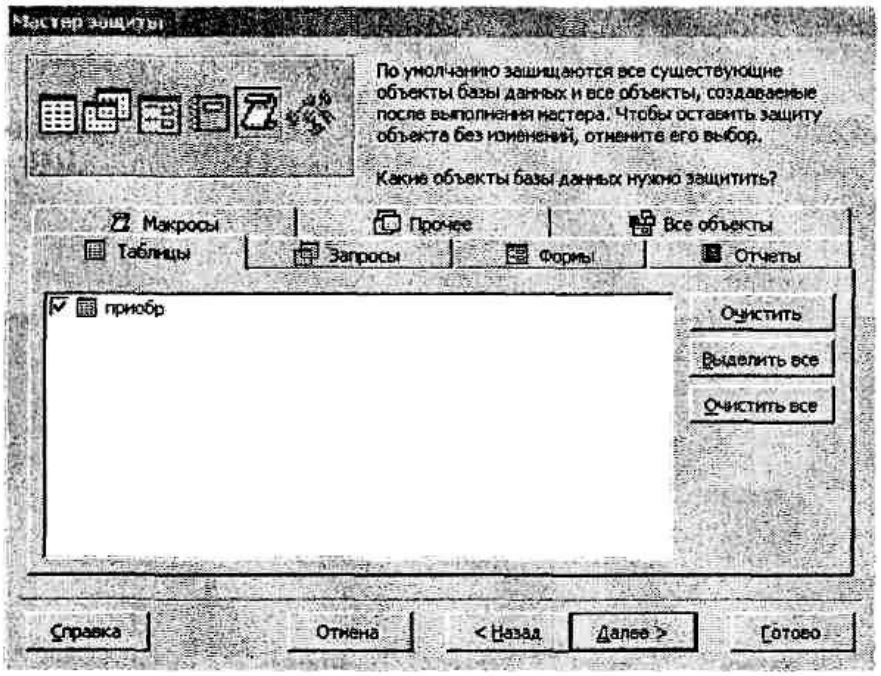


Рис. 10.12. Вид начального окна **Мастер защиты**

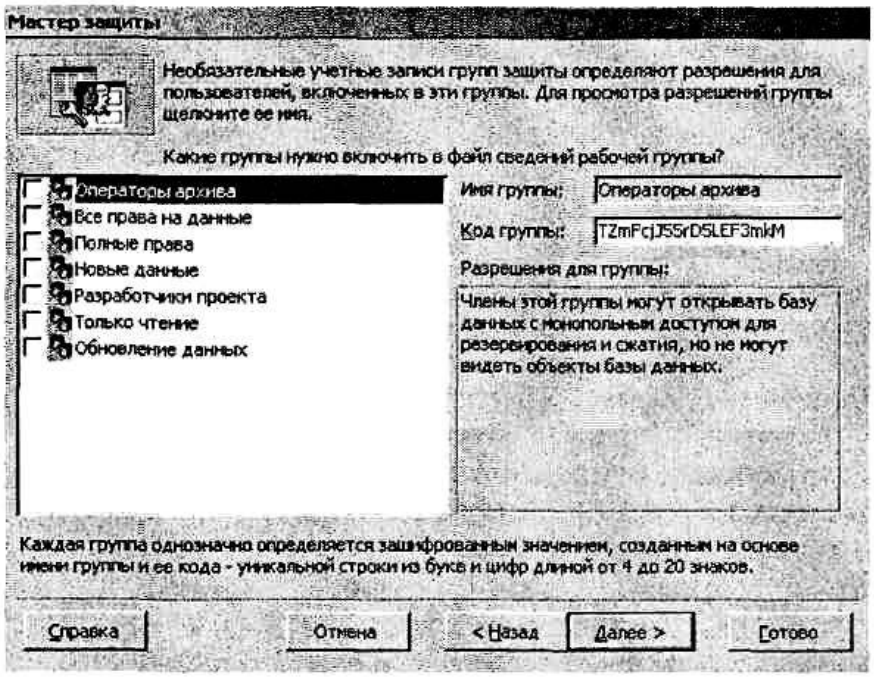


Рис. 10.13. Вид окна **Мастер защиты.** Выбор групп защиты

Пользователь, создавший таблицу, запрос, форму, отчет или макрос, является владельцем этого объекта.

Средства защиты БД

Все те средства защиты БД, которые были перечислены в разд. 10.6.2, могут быть реализованы при использовании Microsoft Access.

Шифрование БД. Для того чтобы зашифровать БД, необходимо запустить Microsoft Access без открытия базы данных. При работе с сетевой базой данных следует убедиться, что все пользователи закрыли базу данных.

В меню **Сервис** нужно выбрать команду **Защита** и подкоманду **Шифровать/дешифровать**, затем - указать имя базы данных, которую требуется зашифровать или дешифровать, и щелкнуть по кнопке **ОК**.

Защита паролем. В Microsoft Access используются три типа паролей: пароли баз данных, пароли учетных записей пользователей и пароли Microsoft Visual Basic для приложений (VBA). Первые два вида паролей обеспечивают безопасность данных, а третий - безопасность программ на языке VBA (предотвращает редактирование, вырезание, вставку, копирование, экспорт и удаление программы несанкционированными пользователями).

Выбранный тип парольной защиты определяет уровень доступа пользователей к базе данных и содержащимся в ней объектам.

Пароли баз данных. Определение пароля базы данных является простейшим средством защиты от открытия базы данных несанкционированным пользователем. Если такой пароль установлен, то от каждого пользователя, открывающего базу данных, требуется ввод этого пароля. Однако после открытия базы данных других средств защиты при этом не имеется, если дополнительно не определена защита на уровне пользователей.

Чтобы задать пароль базы данных, следует выбрать позицию меню **Сервис/Защита/Задать пароль базы данных**. Для того чтобы иметь возможность задать или отменить пароль базы данных, БД должна быть открыта в монопольном режиме (рис. 10.14)

По умолчанию Access открывает базу данных с общим доступом. Для того чтобы использовать базу данных с монопольным доступом, пользователь, который хочет это сделать, должен иметь на это разрешение.

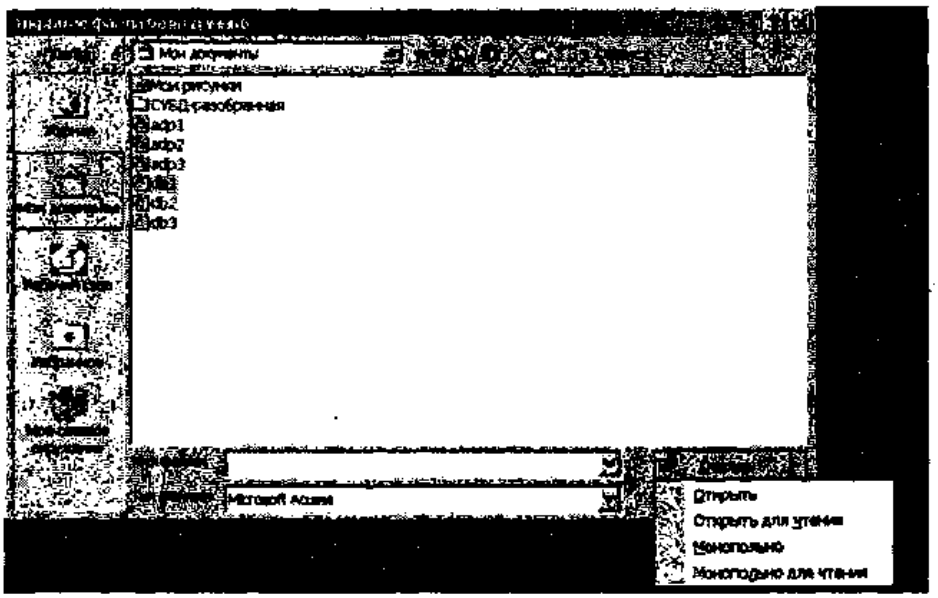


Рис. 10.14. Открытие файла базы данных в монопольном режиме

Пароли учетных записей пользователей. Для того чтобы воспользоваться данным видом защиты, необходимо предварительно создать учетные записи пользователей и групп пользователей.

При использовании способа защиты базы данных на уровне пользователей следует открыть защищаемую базу данных в монопольном режиме (пользователь должен иметь на это соответствующее право), затем - использовать команду Защита в меню Сервис и выбрать позицию «Задать пароль базы данных».

Этот способ достаточно надежен (Microsoft Access шифрует пароль, поэтому к нему нет доступа при непосредственном чтении файла базы данных).

На это следует обратить внимание

- Работа в распределенной многопользовательской среде является в настоящее время преобладающей технологией организации ИС.
- Существует много разновидностей РБД. Особенности каждого класса накладывают отпечаток на процесс проектирования и эксплуатации системы.
- Проектирование РБнД включает в себя все вопросы, присущие проектированию обычных (нераспределенных) систем, и дополнительные вопросы, обусловленные классом создаваемой системы и выбранной технологией.
- Следует уделять серьезное внимание выбору архитектуры РБнД и технологий в рамках выбранной архитектуры, поскольку это решение оказывает большое влияние на показатели надежности, стоимости проектирования и эксплуатации, времени реакции системы.

Контрольные вопросы

1.Что называется распределенным банком данных (РБнД)?

2.Что называется распределенной базой данных»(РБД)?

3.Что понимается под системой управления распределенной базой данных?

4.Что называется сервером баз данных?

5.Что такое SQL-сервер?

6.Какие SQL-серверы вы знаете?

7.Какие требования предъявляются к распределенным банкам данных?

8.Какие классы РБнД вы знаете? Дайте характеристику каждого класса.

9.Назовите преимущества и недостатки централизованных, децентрализованных и распределенных БнД.

10.Какие факторы оказывают влияние на выбор способа организации БД?

11.В чем разница между «файл-серверной» и «клиент-серверной» архитектурой РБнД?

12.Дайте понятие транзакции.

13.Какие требования предъявляются к транзакциям?

14.Какие разновидности транзакций вы знаете?

15.Назовите проблемы, связанные с параллелизмом, и объясните, чем они вызваны.

16.Что такое блокировка? Для каких целей и когда используются блокировки?

17.Какие разновидности блокировок вы знаете?

18.В чем суть разделяемого и исключительного режимов доступа к информации? Когда используется каждый из этих режимов?

19.Что понимается под уровнем изоляции в SQL? Какие уровни изоляции определены в стандарте SQL-92?

20.Что такое хранимые процедуры и для каких целей они используются?

21.Что такое триггеры и для каких целей они используются?

22.В чем суть технологии тиражирования?

23.Что называется синхронизацией?

24.Каковы преимущества и недостатки тиражирования?

25.Какие разновидности тиражирования вы знаете?

26.В чем заключаются особенности обеспечения целостности в РБД?

27.Какие способы защиты данных вы знаете?

28.Как и зачем описываются пользователи, группы пользователей и задаются их права?

29.В чем заключаются особенности работы в распределенной среде при использовании СУБД Access?

ПРИЛОЖЕНИЯ

1. Основные понятия реляционной модели данных

1. Информационные единицы.

1.1. *Реляционная база данных* — совокупность взаимосвязанных плоских таблиц.

1.2. *Таблица* - основная структурная единица реляционной базы данных, представляющая собой подмножество декартова произведения доменов. В реляционной теории используется термин «отношение». Часто эти термины используются как синонимы. Иногда они различаются: таблица считается способом хранения (отображения) отношения.

1.3. Строки таблицы называются кортежами. Кортеж представляет собой совокупность атрибутов (a_1, ..., a_i, a_n). В реляционных СУБД понятию «кортеж» чаще всего соответствует термин «запись».

1.4. Наименьшей информационной единицей является *атрибут*. Каждый атрибут имеет имя (A_i) и значение (a_i). В реляционных СУБД понятию «атрибут» чаще всего соответствует термин «поле».

1.5. *Домен* — множество значений данных какого-либо типа. Каждый атрибут может быть определен на каком-либо одном домене. На одном домене может быть определено несколько атрибутов.

2. Ключи.

2.1. *Ключ* - атрибут или совокупность атрибутов, однозначно определяющая кортеж. Ключ, состоящий из нескольких атрибутов, называется составным.

Ключ обладает следующими свойствами:

а) уникальность - не может быть повторяющихся значений ключа в таблице;

б) неизбыточность - удаление любого атрибута из составного ключа должно привести к потере его уникальности;

в) элементы ключа не могут иметь неопределенные значения.

2.2. В отношении несколько атрибутов или совокупностей атрибутов могут претендовать на роль ключа. В этом случае каждый из них называется вероятным (альтернативным) ключом. Один из них должен быть выбран в качестве первичного ключа.

2.3. Атрибут (или совокупность атрибутов), который в данном отношении ключом не является (но может входить в состав составного ключа), а в другом отношении является первичным ключом, называется внешним ключом.

3. Связи.

Связи в реляционной базе данных устанавливаются от ключа к внешнему ключу.

2. Сквозной пример использования ER-моделирования для проектирования БД

На рис. П.1 изображен фрагмент ER-модели предметной области в нотации базовой модели.

Вариант структуры реляционной базы данных, построенный по алгоритму, изложенному в главе 3, на основе ER-модели, приведен ниже.

ПРЕПОДАВАТЕЛЬ (Таб_ном_преподавателя, ФИО_преподавателя, Дата_рождения, Должность, Уч_степень, Адрес, Условия_сотрудничества).

ПРЕПОДАВАТЕЛЬ ШТАТНЫЙ (Таб_ном_преподавателя, Дата_ зачисления_в_штат).

ПРЕПОДАВАТЕЛЬ ВНЕШТАТНЫЙ (Таб_ном_преподавателя, Место_основной_работы, Должность_по_основной_работе).

ДЕТИ (Таб_ном_преподавателя, ФИО_ребенка).

ЗН_ИН_ЯЗ (Таб_ном_преподавателя, Ин_язык).

ПРЕДМЕТ (Код_предмета, Наименование_предмета_полное, Наименование_предмета_краткое).

ГРУППА (Номер_группы, Вид_группы).

СТУДЕНТ (Ном_зачетки, ФИО_студента, Номер_группы).

СДАЧА_ЭКЗАМЕНА (Таб_ном_преподавателя, Ном_зачетки, Код_предмета, Дата_сдачи, оценка).

Тот же фрагмент ER-модели в нотации IDEF1X, построенный с использованием ERWin, приведен на рис. П.2.

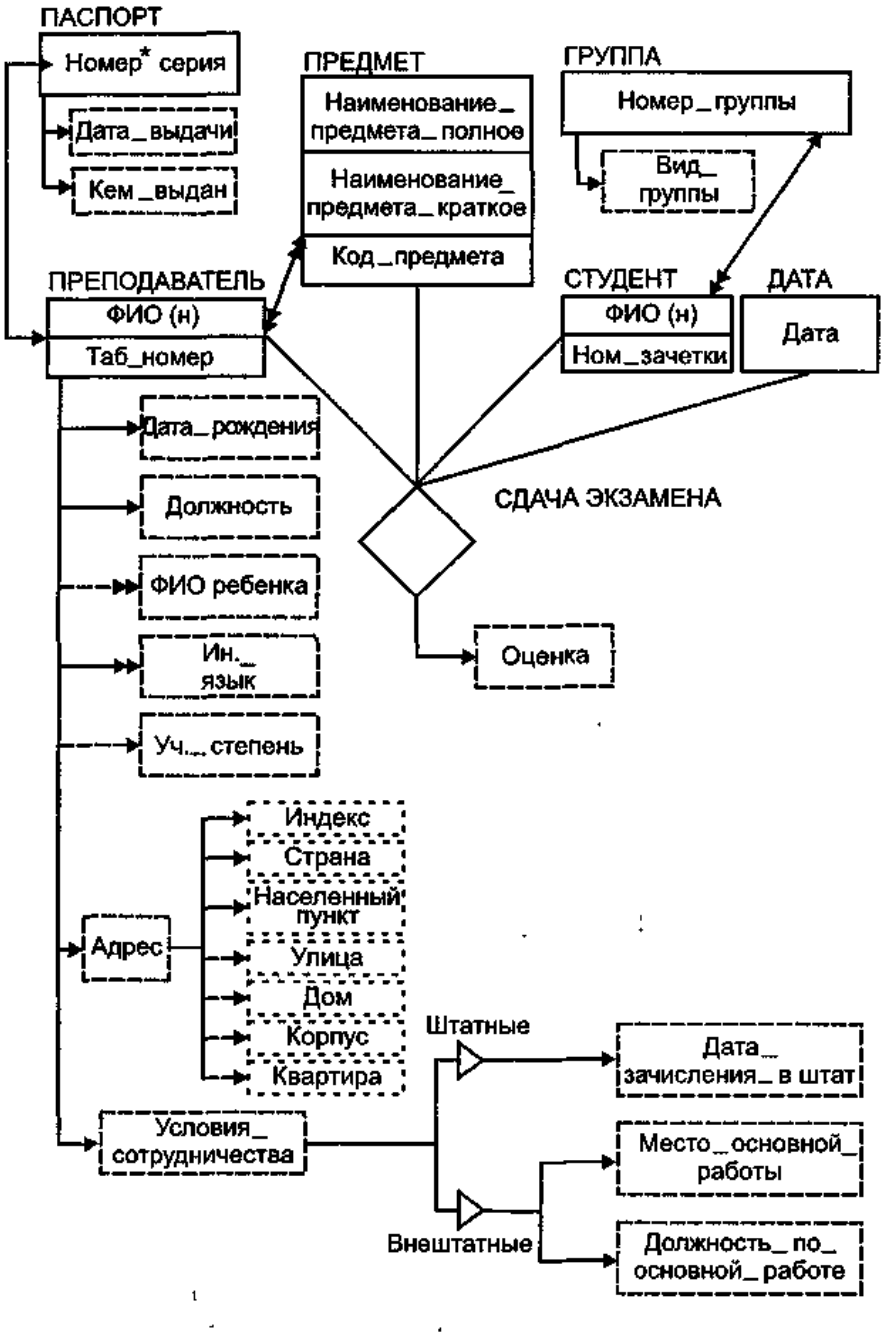


Рис. П.1. Фрагмент ER-модели предметной области (нотация базовой модели)

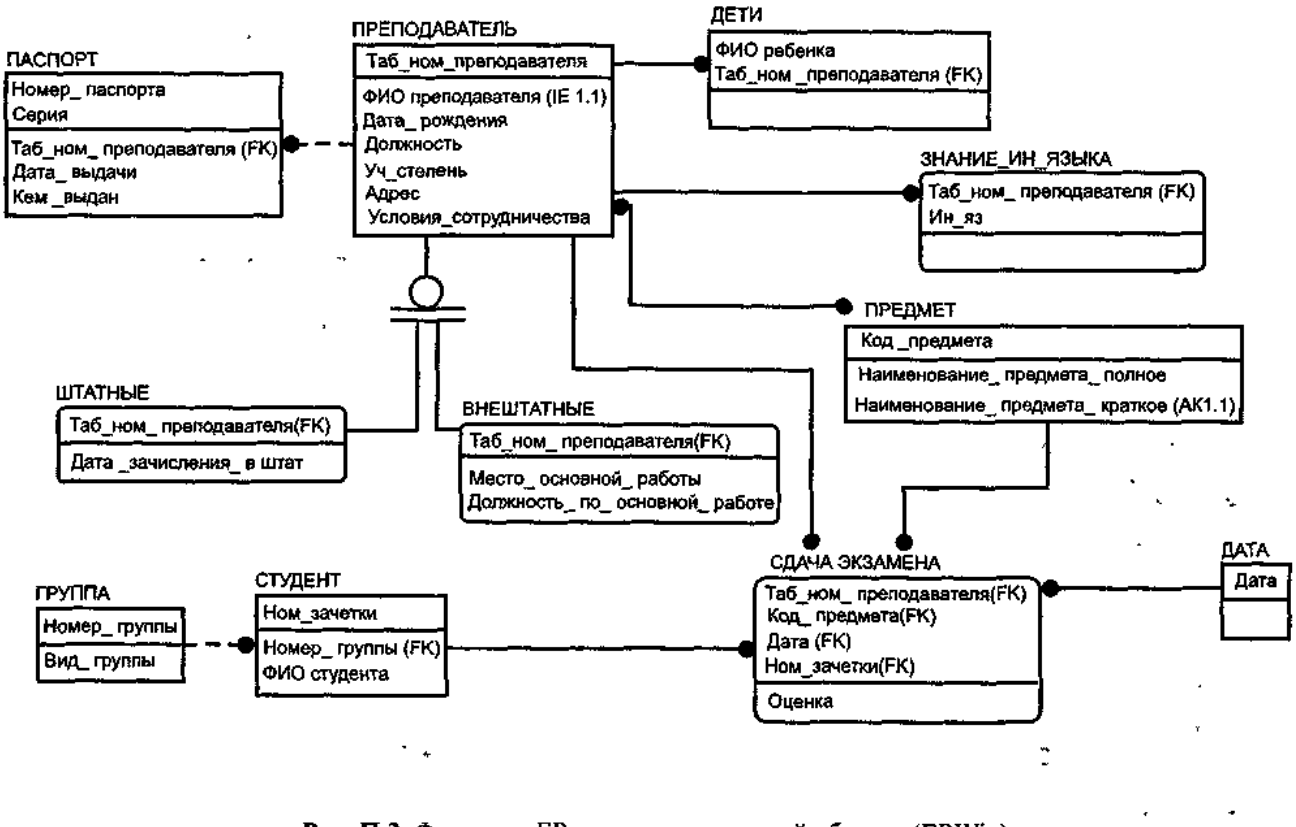


Рис. П.2. Фрагмент ER-модели предметной области (ERWin)

На рис. П2 приведен фрагмент ER-модели для абстрактного учебного заведения. На рисунке продемонстрировано применение всех изобразительных средств, которые имеются в нотации базовой ER-модели. Изображены все виды объектов - простые объекты (ПРЕДМЕТ, ПАСПОРТ, ГРУППА, СТУДЕНТ, ДАТА), обобщенный объект ПРЕПОДАВАТЕЛЬ, агрегированный объект СДАЧА_ЭКЗАМЕНА. Также использованы все виды связей (1:1, 1:M и M: M). В данном примере предполагается, что студент может учиться только в одной группе. Связь M: M между объектами ПРЕПОДАВАТЕЛЬ и ПРЕДМЕТ означает, что преподаватель может владеть несколькими предметами, а каждый предмет могут вести несколько преподавателей.

При изображении объекта ПРЕПОДАВАТЕЛЬ использованы все разновидности свойств. «Должность» изображена как единичное свойство. Это означает, что в данном учебном заведении внутреннее совместительство не разрешено. Если речь идет только о профессорско-преподавательском составе, то решение сделать «Должность» свойством, а не отдельным объектом вполне оправданно. При построении реальных моделей для других предметных областей может возникнуть необходимость изобразить «Должность» как объект.

В данном фрагменте «Дату» можно было бы и не выделять в отдельный объект, сделать свойством агрегированного объекта СДАЧА ЭКЗАМЕНА. В реальной ситуации, когда рисуется общая модель предметной области, с «Датой» бывает связано несколько разных объектов, и в этом случае выделение ее в виде отдельного объекта сократит число элементов в ER-модели, так

объект «Дата» в модель вводится один раз, а соответствующее свойство придется вводить для каждого объекта, где оно встречается.

ГЛОССАРИЙ

2PC - см. Фиксация двухфазная

ER-модель - модель предметной области, отражающая объекты и связи между ними (синонимы: модель «сущность-связь», модель «объект-свойство-связь», концептуальная модель)

SQL-сервер - собирательный термин, относящийся ко всем серверам баз данных, основанных на SQL

Администратор банка данных (АБД) - группа специалистов, обеспечивающих создание, функционирование и развитие БнД

База данных - поименованная совокупность взаимосвязанных данных, находящихся под управлением СУБД

База данных иерархическая - структурированная БД, реализующая иерархическую модель данных

База данных интегрированная - централизованная база данных, предназначенная для многопользовательского обращения

База данных источника - база данных, расположенная на одном из узлов вычислительной сети, после внесения некоторых изменений, в которую возникает необходимость отразить эти изменения на других узлах сети

База данных коммерческая - база данных, предназначенная для продажи

База данных неструктурированная - БД, не требующая предварительного проектирования и описания структуры БД

База данных персональная - база данных, предназначенная для локального использования одним пользователем

База данных приемника - база данных, на которую распространяются изменения в базе данных источника

База данных распределенная (DDB - Distributed DataBase) - совокупность множества взаимосвязанных баз данных, распределенных в компьютерной сети

База данных реляционная - структурированная БД, реализующая реляционную модель данных

База данных сетевая - структурированная БД, реализующая сетевую модель данных

База данных структурированная - БД, требующая предварительного проектирования и описания структуры БД

Банк данных - система специальным образом организованных данных (баз данных), программных, технических, языковых, организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных

Банк данных коммерческий - банк данных, основной целью создания которого является получение прибыли от информационной деятельности.

Банк данных распределенный (РБнД) - БнД, в котором распределен хотя бы какой-либо один компонент БнД (не обязательно БД)

Блокировка - запрещение некоторых операций над данными (чаще - корректировки информации), если их обрабатывает (корректирует) другой пользователь

Владелец информационных ресурсов, информационных систем, технологий и средств их обеспечения - субъект, осуществляющий владение и пользование указанными объектами и реализующий полномочия распоряжения в пределах, установленных законом

Группа пользователей - пользователи, наделенные одинаковым набором привилегий

Идентификатор - уникальное имя объекта

Источник - узел, содержащий базу данных источника

Источник информации - субъект, который порождает информацию

Категории - объекты, составляющие обобщенный объект

Класс объектов - совокупность объектов, обладающих одинаковым набором свойств

Ключ - атрибут или совокупность атрибутов, однозначно идентифицирующих кортеж

Ключ внешний - атрибут или группа атрибутов, которая в рассматриваемом отношении не является ключом, а в другом отношении является первичным ключом

Метаинформация - информация, описывающая базу данных, а также другие части БнД

Модель «объект-свойство-связь» - ER-модель (синоним)

Модель «сущность-связь» - ER-модель (синоним)

Модель внешняя - логическая структура БД с точки зрения конкретного пользователя (синонимы: подсхема, «взгляд»/представление/view)

Модель даталогическая (datalogical) (ДЛМ) - логическая модель БД в терминах конкретной СУБД; отображение логических связей между элементами данных

Модель инфологическая (ИЛМ) - формализованное СУБД-независимое описание предметной области (синоним - концептуальная модель)

Модель концептуальная - см. Модель инфологическая

Модель физическая — описание способа хранения данных в запоминающей среде

Нотация - система обозначений, принятая в конкретной модели

Область предметная - часть реального мира, представляющая интерес для данного исследования

Объект - сущность (синоним)

Объект агрегированный - объект, отражающий объединение других объектов при совершении какого-либо действия (соответствует какому-либо процессу, в который оказываются вовлеченными другие объекты)

Объект обобщенный - объект, включающий подклассы (отражает наличие связи «род-вид» между объектами предметной области)

Объект простой - объект, рассматриваемый в данном исследовании как неделимый

Объект сложный - объект, объединяющий другие объекты, простые или сложные, также отображаемые в информационной системе

Объект составной - объект, включающий в себя в качестве составляющих другие объекты (соответствует отображению отношения «целое-часть» между объектами ПО)

Ограничения целостности - набор специальных предложений, обеспечивающих целостность данных; утверждения о допустимых значениях отдельных информационных единиц и связях между ними

Организационно-методические средства банка данных - инструкции, методические и регламентирующие материалы, предназначенные для пользователей разных категорий, взаимодействующих с банком данных

Подсхема - описание внешней модели

Пользователь - лицо или группа лиц, взаимодействующих с банком данных в процессе его создания и функционирования

Пользователь конечный - пользователи, для нужд которых создается банк данных

Потребитель информации (пользователь) - субъект, обращающийся к информационной системе или посреднику за получением необходимой ему информации и пользующийся ею

Привилегия - право пользователей на выполнение определенных операций над объектами данных некоторого типа

Приемник - узел, содержащий базу данных приемника

Продукты информационные - информационные ресурсы и программы для их обработки

Проектирование даталогическое - этап проектирования БД, включающий работы по созданию ДЛМ

Проектирование инфологическое - этап проектирования БД, включающий работы по созданию ИЛМ

Проектирование физическое - этап проектирования БД, включающий работы по созданию физической модели БД

Процедура хранимая - процедура, хранимая в оттранслированном виде на сервере

Публикация - совокупность данных, которые могут подвергаться тиражированию

Разрешение конфликтов при тиражировании - процесс согласования параллельных изменений в нескольких тиражируемых копиях

Резервирование (архивирование) - создание резервных копий файлов БД

Реорганизация - изменение физического расположения данных на носителе

Реплика - копия базы данных, являющаяся членом набора других копий, которые могут быть синхронизированы между собой

Репликация - см. Тиражирование

Репозиторий - централизованная база данных проекта (см. Словарь данных)

Ресурсы информационные - отдельные документы и отдельные массивы документов, документы и массивы документов в информационных системах (библиотеках, архивах, фондах, банках данных, других информационных системах)

Свойство - характеристика, описывающая состояние каждой сущности

Связь (Relationship) - ассоциация между сущностями, при которой каждый экземпляр одной сущности ассоциирован с произвольным (в том числе нулевым) количеством экземпляров другой сущности

Сеанс (sessions) - работа интерактивного пользователя с системой, выполняемая в промежутке между подключением его к системе и отключением от системы

Сервер баз данных - СУБД, основанная на архитектуре «клиент-сервер»

Сервер подписки - поддерживает базы данных, с которыми имеют дело конечные пользователи информации. Эти БД принимают публикуемые данные

Сервер публикации - сервер, который предоставляет данные для тиражирования

Сервер репликаций - сервер, хранящий базу данных репликаций. В БД репликаций хранятся все изменения публикуемых данных, и по мере необходимости (по плану диспетчирования или другим настройкам) сервер пересылает данные серверам подписки

Сериализуемость - критерий корректности для управления одновременным доступом, который требует, чтобы эффект множества одновременно выполняемых транзакций был эквивалентен эффекту от их последовательного выполнения при каком-либо упорядочении

Синхронизация данных - процесс обновления реплик (в распределенных системах баз данных с тиражированием), обеспечивающий тождественность дублирующихся данных

Система управления распределенной базой данных - программная система, которая позволяет управлять базой данных таким образом, чтобы ее распределенность была прозрачна для пользователей

Словарь данных - централизованное хранилище метаинформации

Снимок базы данных моментальный (Snapshot) - состояние базы данных в целом или ее фрагмента на момент получения снимка

Собственник информационных ресурсов, информационных систем, технологий и средств их обеспечения - субъект, в полном объеме реализующий полномочия владения, пользования, распоряжения указанными объектами

Создание реплики - создание копий баз данных или объектов, допускающих синхронизацию

Статья - наименьший возможный элемент публикации

СУБД - совокупность программных и языковых средств, предназначенных для создания баз данных, поддержания их в актуальном состоянии и организации доступа к ним различных пользователей в условиях принятой технологии данных

СУБД замкнутые - СУБД, имеющие собственные языки общения с пользователями БнД

СУБД иерархическая - СУБД, поддерживающая иерархическую модель данных

СУБД открытые - СУБД, в которых для обращения к базам данных используются универсальные языки программирования

СУБД реляционная - СУБД, поддерживающая реляционную модель данных

СУБД сетевая - СУБД, поддерживающая сетевую модель данных

Сущность - любой реальный либо абстрактный объект, представляющий интерес для рассматриваемой предметной области

Сущность, зависимая от идентификации, - сущность, идентификация которой зависит от идентификации других объектов

Схема - описание логической структуры базы данных

Схема хранения - описание физической структуры базы данных

Технология информационная - совокупность методов, навыков применения всего многообразия вычислительной техники и средств связи в области сбора, обработки, хранения и передачи информации

Тиражирование - используемая в РБнД технология, предусматривающая поддержку копий всей БД или ее фрагментов в нескольких узлах сети

Тиражирование апериодическое - тиражирование, момент выполнения которого определяется каким-либо событием

Тиражирование асинхронное (тиражирование с промежуточным хранением) - схема тиражирования, при которой обновление всех копий баз данных может проводиться неодновременно

Тиражирование периодическое - тиражирование, выполняемое через заданные интервалы времени

Тиражирование синхронное - схема тиражирования, при которой предполагается завершение транзакции только после успешной модификации всех копий

Транзакция - законченная совокупность действий над БД, которая переводит БД из одного целостного в логическом смысле состояния в другое целостное состояние

Триггер - специфицированное в схеме базы данных действие, которое активизируется при наступлении указанного события

Тупик - ситуация, когда множество транзакций образует цикл, ожидая снятия блокировок, установленных другими транзакциями из этого множества

Услуга информационная - услуга по предоставлению в распоряжение пользователя информационных продуктов или самой требуемой информации

Фиксация двухфазная (2PC) - механизм обеспечения целостности при выполнении запроса в распределенной БД, при котором основная система связывается с подчиненными базами данных и одновременно вносит в них изменения, блокируя соответствующие записи. Если одна из таких БД недоступна, изменения не выполняются

Фрагментация - разбиение единой логической БД по каким-либо признакам на составные части (фрагменты), хранящиеся в разных узлах сети

Целостность — актуальность и непротиворечивость базы данных, ее защищенность от разрушения и несанкционированного изменения

Шлюзы (gateways) - программные средства, позволяющие совместно обрабатывать информацию, различающуюся по форме представления

ЛИТЕРАТУРА

1. *Боггс У., Боггс М.* UML и Rational Rose. - М.: ЛОРИ, 2000. /
2. *Буч Г.* Объектно-ориентированное проектирование с примерами применения. - М.: Радио и связь, 1992.
3. *Вендров А.М.* CASE-технологии. Современные методы и средства проектирования информационных систем. - М.: Финансы и статистика, 2000.
4. *Вендров А.М.* Практикум по проектированию программного обеспечения экономических информационных систем. - М.: Финансы и статистика, 2002.
5. *Вендров А.М.* Проектирование программного обеспечения экономических информационных систем: Учебник. - М.: Финансы и статистика, 2004.
6. *Вескес Дж., Гандерлоу М., Чипмен М.* Access и SQL Server. Руководство разработчика: Пер. с англ. - М.: ЛОРИ, 1997.
7. *Гейн К., Сарсон Т.* Структурный системный анализ: средства и методы: Пер. с англ. - М.: Эйтекс, 1993.
8. *Глушаков С.В.* Базы данных. - М.: Фолио, 2001.
9. *Горев А., Ахаян Р., Макашарипов С.* Эффективная работа с СУБД. - СПб.: Питер, 1997.
10. *Грабер М.* SQL.-М.: ЛОРИ, 2001.
11. *Грабер М.* SQL. Справочное руководство. - М.: ЛОРИ, 1997.
12. *Грабер М.* Введение в SQL.: Пер. с англ. - М.: ЛОРИ, 1996.
13. *Грачев А.Ю.* Введение в СУБД Informix. - М.: ДИАЛОГ-МИФИ, 2000.
14. *Дейт К. Дж.* Введение в системы баз данных. - 6-е изд.: Пер. с англ. - СПб.: Издательский дом «Вильяме», 2000.
15. *Джексон Г.* Проектирование реляционных баз данных для использования с микроЭВМ: Пер. с англ. - М.: Мир, 1991.
16. *Джонсон Р.* и др. Приемы объектно-ориентированного проектирования. - СПб.: Питер, 2001.
17. *Диго С. М.* Проектирование баз данных: Учебник. - М.: Финансы и статистика, 1988.
18. *Диго С. М.* Проектирование и использование баз данных: Учебник; -\ М.: Финансы и статистика, 1995. \
19. *Дюк В., Самойленко А.* Data mining: Учебный курс. - СПб: Питер, 2001.
20. *Йордон Э., Аргила С.* Структурные модели в объектно-ориентированном анализе и проектировании. - М.: ЛОРИ, 1999.
21. *Каляное Г.Н.* CASE - структурный системный анализ. - М.: ЛОРИ, 1996.
22. *Каляное Г.Н.* Консалтинг при автоматизации предприятий. - М.: СИНТЕГ, 1998.
23. *Когаловский М.Р.* Энциклопедия технологий баз данных. - М.: Финансы и статистика, 2002.
24. *Кренке Д.* Теория и практика построения баз данных. - СПб: Питер, 2003.

25.*Маклаков С.В.* BPWin и ERWin. CASE-средства разработки информационных систем. - М.: ДИАЛОГ-МИФИ, 1999.

26.*Мацяше Л.* Анализ требований и проектирование систем. Разработка информационных систем с использованием UML: Пер. с англ. - М.: Вильяме, 2002.

27.*Михеев В.Д.* Access 2002 в подлиннике. - СПб: БХВ, 2002.

28.*Мишенин А.И.* Теория экономических информационных систем. - М.: Финансы и статистика, 2003.

29.*Росс Д.Т.* Структурный анализ (SA): язык для передачи понимания. - М.: Мир, 1984.

30.*Савицкий Н.И.* Технологии организации, хранения и обработки данных: Учеб. пособие. - М.: Инфра-М, 2001.

31.*Саймон А.Р.* Стратегические технологии баз данных: менеджмент на 2000 год. - М.: Финансы и статистика, 1998.

32.*Смирнова Г.Н., Сорокин А.А., Тельное Ю.Ф.* Проектирование экономических информационных систем. - М.: Финансы и статистика, 2003.

33.*Спирли Э.* Корпоративные хранилища данных. Планирование, разработка, реализация. Т. 1: Пер. с англ. - М.: Вильяме, 2001.

34.*Фаулер М.*, *Скотт К.* UML в кратком изложении. Применение стандартного языка объектного моделирования: Пер. с англ. - М.: Мир, 1999.

35.*Фридман А.Л.* Основы объектно-ориентированной разработки программных систем. - М.: Финансы и статистика, 2000. /

36.*Хансен Г., Хансен Дж.* Базы данных. Разработка и управление. / М.: Бином, 2000. /

37.*Хендерсон К.* Delphi 3 и системы клиент/сервер. Руководство/разработчика: Пер. с англ. - Киев: Диалектика, 1997.

38.*Черемных С.В., Семенов И.О., Ручкин В.С.* Структурный анализ систем: IDEF-технологии. - М.: Финансы и статистика, 2003.

39.*Чери С, Готлоб Г., Танка Л.* Логическое программирование и базы данных: Пер. с англ. - М.: Мир, 1992.

40.*Четвериков В.Н., Ревунков Г.И.*, Самохвалов Э. Н. Базы и банки данных. - М.: Высшая школа, 1987.

41.*Шлеер С, Меллор С.* Объектно-ориентированный анализ: моделирование мира в состояниях: Пер. с англ. - Киев: Диалектика, 1993.

42.*Hansen Gary W., Hansen James V.* Database Management and Design: New Jersy, Prentice-Hall, Inc. - 1992.

43.*Martin, James.* Fourth-generation languages. VOL. 1. Priciples. - New Jersy, Prentice-Hall, Inc. -1985.

СОКРАЩЕНИЯ

АБД - администратор базы данных
АБнД - администратор банка данных
АИС - автоматизированная информационная система
БД - база данных
БнД - банк данных
ДЛМ - даталогическая модель базы данных
ИБД - интенсиональная база данных
ИЕ - информационная единица
ИЛМ - инфологическая модель предметной области
ИО - идентификатор объекта
ИС - информационная система
КБнД - коммерческий банк данных
КМ - концептуальная модель
ООБД - объектно-ориентированные базы данных
ОС - операционная система
ПО - предметная область
ПП - программный продукт
ППП - пакет прикладных программ
РБД - распределенные базы данных
РБнД - распределенные банки данных
РСБД - расширяемая система баз данных
СВТ - средства вычислительной техники
СЕИ - составная единица информации
СОД - система обработки данных
СООБД - система объектно-ориентированных баз данных
СППР - система поддержки принятия решений
СУБД - система управления базами данных
ЭБД - экстенсиональная база данных
ЯЗ - язык запросов
ЯМД - язык манипулирования данными
ЯОД - язык описания данных
ЯОПС - язык описания подсхем
ЯОС - язык описания схем
ЯОСХ - язык описания схем хранения
ЯОХД - язык описания хранимых данных
2PC (Two-Phase Commit) - двухфазная фиксация - метод синхронного внесения изменений в несколько БД
ACID (Atomicity, Consistency, Isolation, Durability) - атомарность, непротиворечивость, изолированность, долговечность
AL (Access Layer) - уровень доступа к ресурсам
API (Aplication Programming Interface) - интерфейс прикладного программирования

BL (Business Layer) - уровень бизнес-логики

BPM (Business Process Model) - модель бизнес-процесса

BPR (Business Process Re-engineering) - бизнес-процесс реинжиниринг (БПР)

CASE (Computer-Aided Software Engineering,) - система автоматизированной разработки программ

CODASYL (Conference on DAta SYstems Languages) - Конференция по языкам информационных систем

CORBA (Common Object Request Broker Architecture) - единая архитектура программы-брокера Объектных Запросов. Составная Часть (OMA); другой перевод - общий брокер объектных заявок

DBA (DataBase Administration) - администрирование базы данных DBMS (DataBase Management System) - система управления базами данных (СУБД)

DCL (Data Control Language) - язык управления данными

DDB (Distributed DataBase) - распределенная база данных (РБД)

DDE (Dynamic Data Exchange) - динамический обмен данными

DDL (Data Definition Language) - язык определения данных

DFD (Data Flow Diagrams) - диаграммы потоков данных

DSS (Decision Support Systems) - системы поддержки принятия решения (СППР)

EIS (Executive Information System) - исполнительная информационная система (информационная система руководителя)

ERD (Entity-Relationship Diagrams) - диаграммы «сущность-связь»;

GUI (Graphical User Interface) - графический интерфейс пользователя (ГИП)

HTML (Hypertext Markup Language) - язык гипертекстовой разметки

ID API (Integrate Database Application Program Interface) - набор функций для доступа к разным базам данных

IE (Information Engineering) - информационный инжиниринг

IQ (Interactive Query) - интерактивные запросы

MDD (MultiDimensional Database) - многомерные БД

ODBC (Open DataBase Connectivity) - стандарт интероперабельной работы с данными

OLAP(On-Line Analytical Processing) - оперативная аналитическая обработка данных

OLE (Object Linking Embedding) - метод связывания документа-контейнера и документа-сервера, при котором данные сервера сохраняются в том же файле, что и данные контейнера

OLTP (On-Line Transaction Processing) - система оперативной обработки транзакций

PL (Presentation Layer) - презентационный уровень

QBE (Query-By-Example) - запрос по примеру

RAD (Rapid Application Development) - средства быстрой разработки приложений

RDA (Remote Data Access) - доступ к удаленным данным

ROWA (Read-Once/Write-All) - одно чтение, запись во все копии

SQL (Stratured Query Language) - структурированный язык запросов

UML (Unified Modeling Language) - унифицированный язык моделирования

WYSIWYG (What You See Is What You Get) - «что видишь, то и получишь» (режим полного соответствия изображения на экране и распечатки)

XML (Extensible Markup Language) - расширяемый язык разметки