Международный консорциум «Электронный университет»

Московский государственный университет экономики, статистики и информатики

Евразийский открытый институт

С.М. Диго

Базы данных. Проектирование и создание

Учебно-методический комплекс

Москва 2008

УДК 16
ББК 87.4
Д 671

Диго С.М. **БАЗЫ ДАННЫХ. ПРОЕКТИРОВАНИЕ И СОЗДАНИЕ**: Учебно-методический комплекс. – М.: Изд. центр ЕАОИ. 2008. – 171 с.

ISBN 978-5-374-00055-9

ОГЛАВЛЕНИЕ

Цели и задачи дисциплины

Цели изучения дисциплины

Курс «Базы данных» в конкретных областях деятельности преследует несколько целей:

1. показать особенности технологии банков данных как одной из основных новых информационных технологий, с тем чтобы студенты понимали тенденции развития современных информационных технологий, видели их преимущества и недостатки, особенности работы в условиях конкретных технологий в их профессиональной деятельности;
2. сориентировать студентов во множестве современных СУБД и связанных с ними технологий;
3. осветить теоретические и организационно-методических вопросы построения и функционирования систем, основанных на концепции баз данных, в том числе различные методологии моделирования и проектирования баз данных;
4. показать возможности средств автоматизации проектирования БД;
5. показать возможности современных высокоуровневых языков и средств создания приложений;
6. научить практической работе (проектирование, ведение и использование баз данных) в среде выбранных целевых СУБД.

Задачи изучения дисциплины

Задачей изучения дисциплины является научить студентов квалифицированно использовать возможности баз данных. В процессе изучения дисциплины студенты должны:

Иметь представление: об основных понятиях БД, компонентах банков данных, разновидностях банков данных и их особенностях, подходах к построению БД и сферы их применимости.

Знать: особенности реляционной модели и их влияние на проектирование БД, изобразительные средства, используемые в ER-моделировании;языки описания и манипулирования данными разных классов (QBE, SQL, элементы 4GL), технологии организации БД.

Уметь: определить предметную область, спроектировать реляционную базу данных (определить состав каждой таблицы, типы полей, ключ для каждой таблицы), определить ограничения целостности, получать результатные данные в различном виде (ответов на запросы, экранных форм, отчетов).

Глава 1.

Введение в банки данных

1.1. Понятие банка данных

1.1.1. Основные понятия

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»[1].

В данном 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, в этом определении также обозначены и основные *отличительные особенности банков данных:*

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

Использование тех или иных терминов зависит от аспекта рассмотрения изучаемой проблемы. Так, например, в ФЗ «О правовой охране программ для электронных вычислительных машин и баз данных» (№3523-1 от 23.09.92) под базой данных понимается практически любая совокупность данных, которая может быть обработана с помощью ЭВМ. И это оправдано, так как права собственности и иные права не могут зависеть от того, при помощи 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.

[1] Общеотраслевые руководящие материалы по созданию банков данных. – М.: ГКНТ, 1982.

1.1.2. Преимущества БнД

Особенности «банковской» организации данных определяют 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 функций и, как следствие, улучшить всю систему управления предприятием.

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

1.1.3. 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 используются и для удовлетворения непроизводственных информационных потребностей. Конечные пользователи – наиболее многочисленная группа пользователей. Нельзя недооценивать важности этой группы и не понимать специфических особенностей для каждой из категорий конечных пользователей БнД.

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

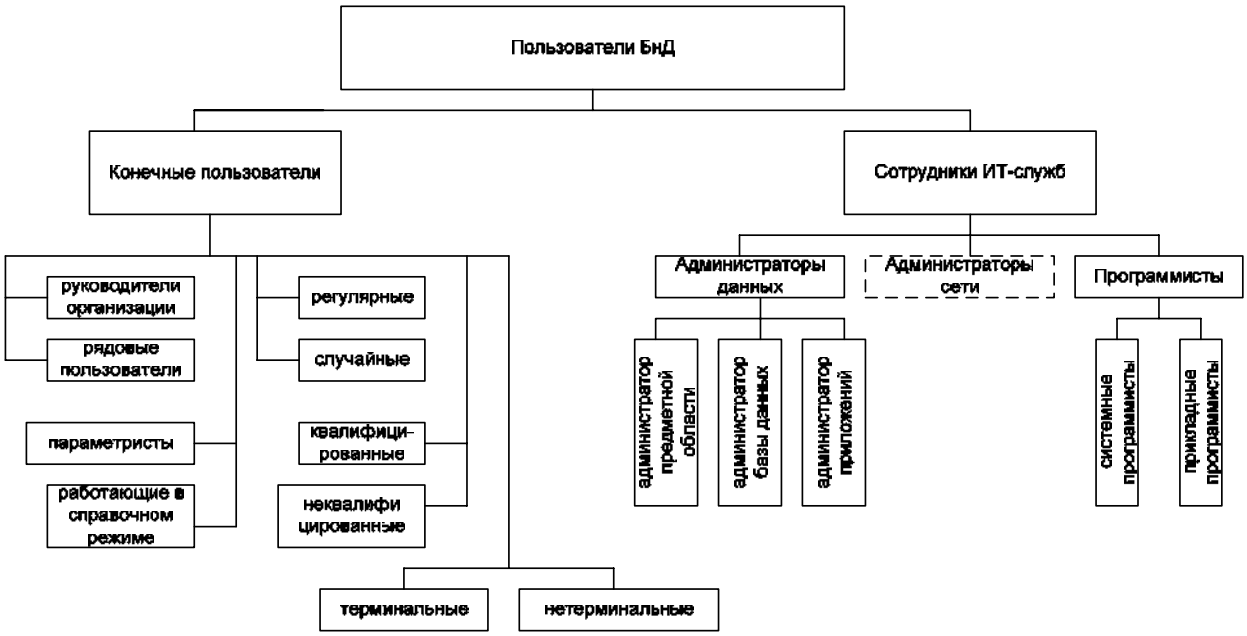


Рис. 1.1. **Пользователи банков данных**

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.

В связи с тем что 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 пользователи-параметристы. Другие могут только использовать хранящуюся в БД информацию.

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

Понятием «конечные пользователи» определяется 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 профессионалов 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.

1.1.4. 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;
- 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 приводит 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 позволяют создавать эффективные БнД.

1.1.5. 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.).

1.1.6. 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.

Тем не менее преимущества БнД значительно превосходят их недостатки. 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. 1.2).

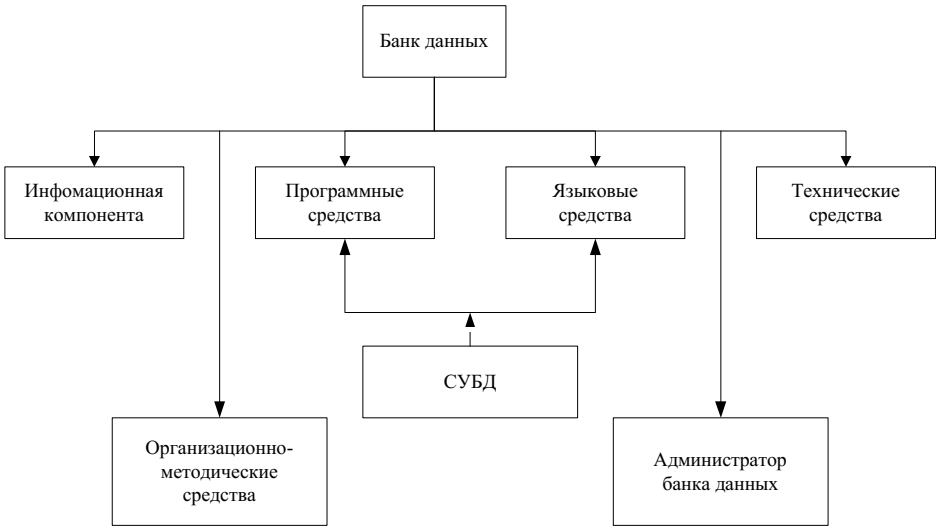


Рис. 1.2. **Компоненты банка данных**

1.2.1. Информационная компонента

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

В настоящее время действует закон «О правовой охране программ для электронных вычислительных машин и баз данных». В этом законе 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» (ст. 1). Учитывая назначение этого закона, вполне естественно, что здесь сделан иной акцент, чем в данном

нами определении БД; определение, используемое в тексте данного Закона, является более широким, чем приведенное нами.

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*. 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***.

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 т.д.

Централизованное хранилище метаинформации называется ***словарем данных***. В литературе используются также термины словарь-справочник, энциклопедия, репозиторий. Роль словарной системы особенно возрастает при использовании средств автоматизированного проектирования информационных систем. Для большинства из них репозиторий является ядром всей системы. Также велика роль репозитория в распределенных системах.

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. 1.3).

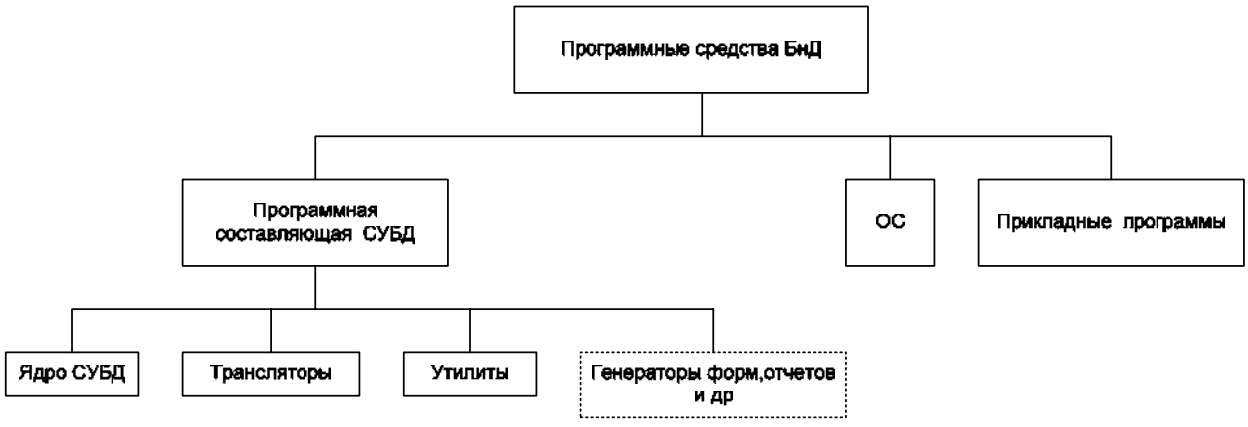


Рис. 1.3. **Программные средства БнД**

Основу программного обеспечения БнД представляют программные компоненты СУБД. 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[1], которые представляют прикладное программное обеспечение БнД.

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.

1.2.3. Языковые средства БнД

Языковые средства СУБД являются важнейшей компонентой банков данных, так как в конечном счете они обеспечивают интерфейс пользователей разных категорий с банком данных. Набор используемых в БнД языковых средств широк и разнообразен, их можно классифицировать по следующим признакам (рис. 1.4).

Рис. 1.4. **Языковые средства СУБД**

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

Широко используемым является деление языковых средств ***по поколениям***. К *первому поколению* относят машинные языки, ко *второму* – символические языки ассемблера, к *третьему* – алгоритмические языки типа PL, COBOL и т.п., которые в 60-е годы назывались языками высокого уровня, но уровень которых гораздо ниже, чем у языков четвертого поколения.

Языки четвертого поколения создавались под девизом: «люди стоят дороже, чем машины»[1]. При их проектировании используются 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 либо 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 позволить легко вносить изменения в имеющиеся приложения.
7. Принцип максимума результата. Языки предоставляют пользователям мощный инструмент для решения разнообразных задач.

Имеются еще и языки *пятого* поколения, к которому относят языки систем искусственного интеллекта.

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 (***функция***) 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 – многие 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 объединены.

Языки манипулирования данными разделяются на две большие группы: *процедурные* и *непроцедурные*. 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 поза-

[1] Martin James. Fourth-generation languages. – Vol. 1. – New Jersy: Prentice-Hall, Inc., 1989.

писную 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: в частности, табличный язык QBE и язык запросов SQL (основан на реляционном исчислении кортежей).

По форме представления различают *аналитические, табличные и графические* языковые средства. Классификация языковых средств по форме представления относится как к языкам описания данных, так и к языкам манипулирования данными. Так, описание таблицы с использованием команды CREATE TABLE языка SQL является примером аналитической формы ЯОД, а описание такой же таблицы в Access и большинстве других настольных СУБД – пример табличной формы описания. В качестве примеров табличной и аналитической формы задания запросов можно привести языки QBE и SQL соответственно.

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

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.

1.2.4. Технические средства БнД

В качестве технических средств для банков данных чаще всего используются универсальные ЭВМ, периферийные средства для ввода информации в базу данных и отображения выводимой информации. Иногда используются дополнительные технические средства для хранения больших объемов данных на внешних носителях. Если банк данных реализуется в сети, то необходимы соответствующие технические средства для обеспечения ее работы.

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

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.

Получила некоторое развитие идея использования так называемых «сетевых компьютеров». Они представляют собой дешевые рабочие станции без дисковых накопителей[1], которые будут работать в сети и использовать и программные средства, и данные, которые находятся 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). Использование сетевых компьютеров обусловлено не столько тем, чтобы сэкономить за счет использования 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;
- 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 легче, что немаловажно для мобильных пользователей, а также дешевле переносных ПК. Ведущие производители СУБД приспосабливают свои крупномасштабные серверные системы для доступа из карманных ПК.

Технические средства БнД не ограничиваются только ЭВМ. Сюда входит весь комплекс технических средств хранения, отображения и передачи информации. 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 не только для организации собственно баз данных, предназначенных для оперативного доступа к ним, но и архивных данных.

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

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] Позднее появились предложения создавать сетевые компьютеры с собственными накопителями.

1.2.6. Администраторы банка данных

Функционирование БнД невозможно без участия специалистов, обеспечивающих создание, функционирование и развитие БнД. Их называют администраторами банка данных (АБД), они считаются 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, выполняемые ими 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.

Связи администратора банка данных. В процессе своей деятельности администратор БнД взаимодействует с другими категориями пользователей банка данных, а также с «внешними» специалистами, не являющимися пользователями БнД (рис. 1.5).

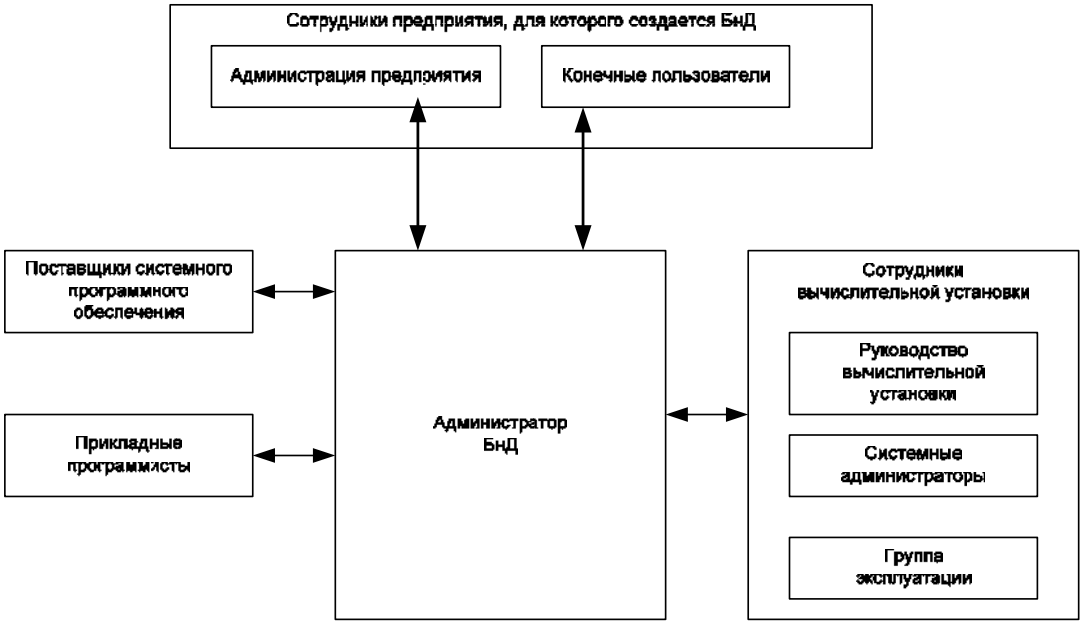


Рис. 1.5. **Взаимодействие администраторов банка данных с другими категориями пользователей**

Прежде всего, если банк данных создается для информационного обслуживания какого-либо предприятия или организации, то 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 проблемах, вызываемых созданием и функционированием БнД.

Так как 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 АБнД на всех этапах жизненного цикла БнД наблюдается с конечными пользователями. Это взаимодействие начинается на начальных стадиях проектирования системы, когда изучаются потребности пользователей, уточняются особенности предметной области, и постоянно поддерживается как на протяжении процесса проектирования, так 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. Сюда относятся простые и одновременно мощные языки запросов, а также средства автоматизации проектирования.

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

Администраторы БнД взаимодействуют и с внешними по отношению к нему группами специалистов, 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.

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 по множеству разных признаков (табл. 1.1).

Таблица 1.1

Классификация БнД

БД	**СУБД**	**К БнД в целом**
По форме представления информации ↓ визуальные ↓ аудио ↓ мультимедиа	*По языкам общения* ↓ открытые ↓ замкнутые ↓ смешанные	*По условиям предоставления услуг* ↓ бесплатные ↓ платные (бесприбыльные коммерческие)
По характеру организации данных ↓ Неструктурированные ↓ частично структурированные ↓ структурированные (по типу используемой модели: иерархические, сетевые, реляционные, смешанные, мультимодельные)	*По числу уровней в архитектуре* ↓ одноуровневые ↓ двухуровневые ↓ трехуровневые	*По характеру преобладающей обработки информации* ↓ OLTP ↓ OLAP
По типу хранимой информации ↓ документальные (библиографические, реферативные, полнотекстовые) ↓ фактографические ↓ лексикографические	*По выполняемым функциям* ↓ информационные ↓ операционные	*По степени доступности* ↓ общедоступные ↓ с ограниченным кругом пользователей
По характеру организации хранения данных и обращения к ним ↓ локальные ↓ общие ↓ распределённые	*По сфере возможного применения* ↓ универсальные ↓ специализированные	*По охвату* ↓ территориальные ↓ временные ↓ ведомственные ↓ проблемные
По способу задания метаинформации ↓ экстенсиональные ↓ интенсиональные	*По «мощности»* ↓ настольные ↓ корпоративные	*По характеру взаимодействия с пользователями* ↓ активные ↓ пассивные
	По ориентации на преобладающую категорию пользователей ↓ для разработчиков ↓ для конечных пользователей	*По форме собственности* ↓ государственные ↓ негосударственные (частные, групповые, личные)

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 *аудио*системы, 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, движущие 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, 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, образуют *запись*, а множество однотипных записей представляют *файл базы данных*. Совокупность взаимосвязанных файлов БД образует *базу данных*.

На рис. 1.6 приведена схема иерархической, а на рис. 1.7 – сетевой модели данных.

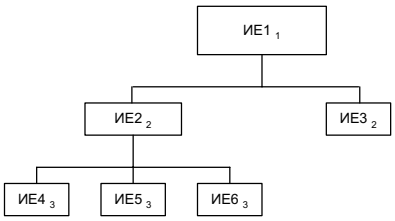


Рис. 1.6. **Схема иерархической модели**

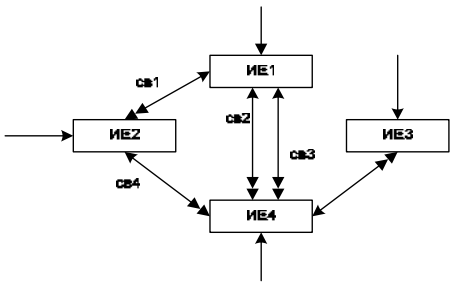


Рис. 1.7. **Схема сетевой модели**

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 вершина, 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. 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.

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 ограничения 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.

Особое 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. 1.8 схематически представлен принцип установления связей в таких системах. Реально, связи устанавливаются не непосредственно с элементами связи, как это изображено на рисунке, а через преобразователь адреса. В системах, построенных на инвертированных файлах, можно передавать связь типа M:M 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.

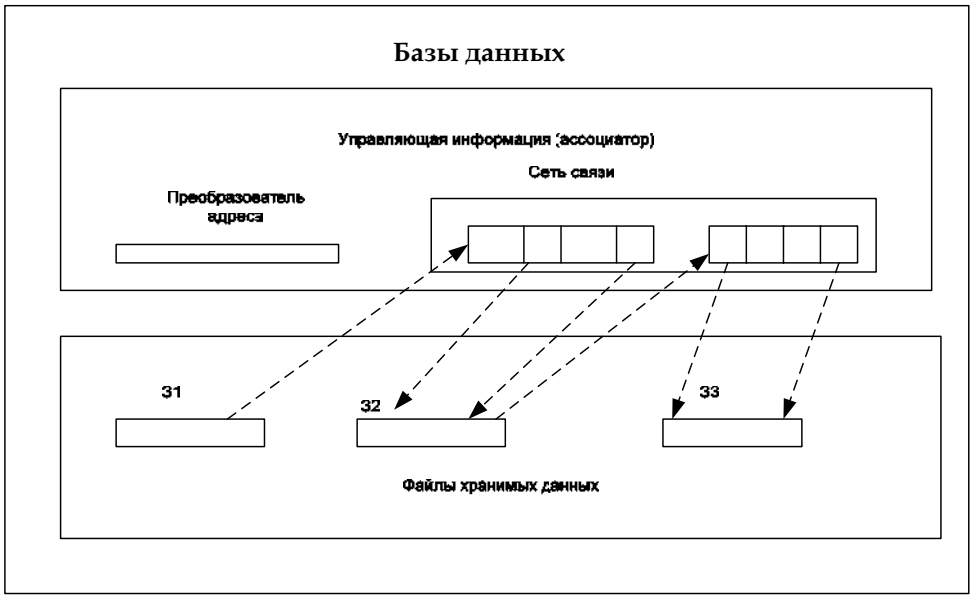


Рис. 1.8. **Схема организации данных в системах, основанных на инвертированных файлах**

Основной информационной единицей в реляционных базах данных является плоская двумерная таблица. Отличительной чертой реляционных моделей является ограничение на внутризаписную структуру: записи имеют линейную структуру и могут содержать только простые поля (рис. 1.9).

Другой особенностью реляционных моделей является то, что связи между записями соответствующих таблиц определяются динамически в момент выполнения запроса. Эти связи определяются по равенству значений соответствующих полей (полей связи), содержащихся в каждой из связанных таблиц.

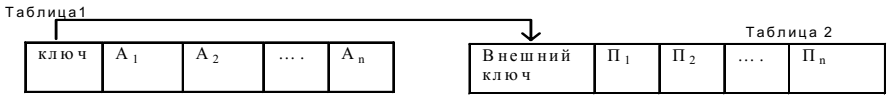


Рис. 1.9. **Схема реляционной модели**

Эти отличительные особенности играют решающую роль при проектировании структуры БД.

Модель данных, кроме особенностей структуры данных, характеризуется также используемыми языками манипулирования данными. Особенностью реляционных моделей является то, что в этих системах должны использоваться теоретико-множественные ЯМД. Восьмидесятые годы были временем интенсивного развития реляционных систем. В 1992 году[1] уровень продаж реляционных СУБД впервые превысил уровень продаж нереляционных СУБД. Но до 90% данных предприятий хранилось к этому моменту в нереляционных базах данных на мэйнфреймах. Преобладание популярности реляционных СУБД сохранилось до настоящего времени.

[1] Первая коммерческая реляционная СУБД была выпущена фирмой Oracle в 1979 году.

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 ему 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, так и 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 систем (ищется документ, а не информация о конкретном объекте, форма документа имеет название, по которому обычно и осуществляется ее поиск), а также специфическими особенностями (документ ищется не с целью извлечь из него информацию, а с целью использования 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, которые оперируют этими данными. Объект включает метод, который является частью определения объекта и запоминается вместе с объектом. В объектных БД 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. 1.10).

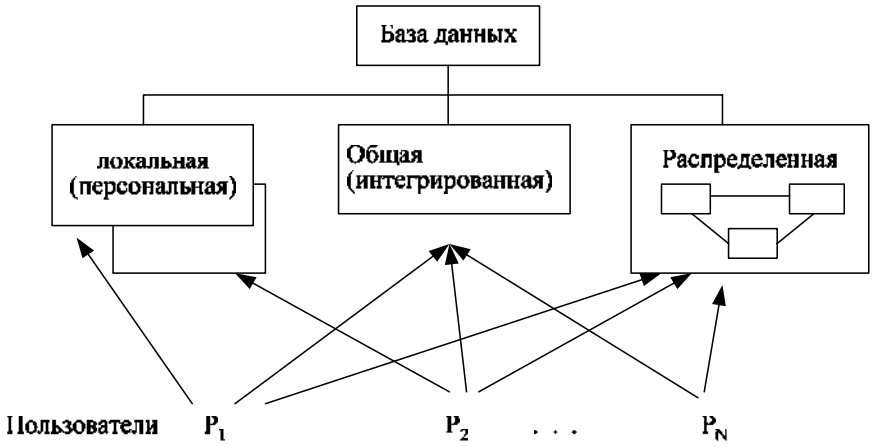


Рис. 1.10. **Классификация БнД по характеру хранения данных и обращения к ним**

Персональная база данных – предназначена 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 этого, 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 случае локальных систем поддержание этой связи не является централизованной, а в случае распределенных БнД – должна поддерживаться СУБД. Технологией, позволяющей совмещать идеи локальной работы и централизованного поддержания единой БД, является технология тиражирования, при которой средства СУБД позволяют тиражировать отдельные части общей БД, локально использовать их, а потом «согласовывать» отдельные фрагменты БД в рамках единой базы данных.

Концепции централизованной и распределенной обработки данных также 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 не распределенной (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 распределена хотя бы одна любая из его компонент.

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

По способу задачи метаинформации различают *экстенсиональные* (ЭБД) и *интенсиональные* БД. Интенсиональная база данных (ИБД) строится с помощью явного хранения данных в БД, как в экстенсиональных 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). Мы можем построить в этой ЭБД вторую таблицу РОДИТЕЛЬ (PARENT), которая содержит поля ФАМИЛИЯ_ИМЯ_ОТЧЕСТВО родителя (FIO) и ИМЯ_РЕБЕНКА (CHILD). С помощью правил мы можем определить, например, отношение ОТЦА (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 выдадутся 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 общем, 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»).

1.3.2. Классификация 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).

На рис. 1.11 сделана попытка совместить терминологию, встречающуюся в разных литературных источниках. В литературе широко используются понятия «внешняя», «концептуальная» и «внутренняя» модель/уровень[1], а также «логический» и «физический»[2] уровень[3], а кроме того «внешняя схема», «подсхема», «схема хранения», просто «схема» и проч. Понятие «схема» с тем или иным уточнением обычно относится к описанию соответствующего уровня описания данных.

---

[1] Мишенин А.И. Теория экономических информационных систем. – М.: Финансы и статистика, 2003.

[2] Во многих современных Case-средствах концептуальной моделью называется ER-модель (ER – Entity – Relationship, сущность – отношение) предметной области, 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 (так как ни о какой «физике» речь здесь не идет).

[3] Хансен Г., Хансен Дж. Базы данных. Разработка и управление. – М.: Бином, 1999.

Рис 1.11. **Классификация СУБД по числу уровней в архитектуре (пример трехуровневой архитектуры)**

Нумерация уровней на рисунке условна, но тем не менее отражает их значимость (внутренняя модель может быть построена только на основе концептуальной; эти два уровня могут быть совмещены, но поддерживаются СУБД всегда; внешний уровень в архитектуре СУБД может отсутствовать).

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*» (табл. 1.2). 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. В связи с этим очевидно, что корпоративные СУБД сложны, дороги, требуют значительных вычислительных ресурсов.

Таблица 1.2

Сравнение «настольных» и «корпоративных» СУБД

Критерий	Настольные	Корпоративные
Простота использования	+	
Стоимость программного обеспечения	+	
Стоимость эксплуатации	+	
Функциональные возможности, в т.ч.: – возможности администрирования – возможности работы 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 них. 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, Informix, Sybase, MS SQL Server, Progress, DB2 и некоторые другие.

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 обладать другими возможностями, позволяющими создавать 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.

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 БД 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*** (табл. 1.3). 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 является сравнительно новым и в разных литературных источниках трактуется иногда по-разному. OLAP 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). «Складам данных» присущ многомерный характер запросов. Огромные объемы данных, сложность структуры как данных, так и запросов требует 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 одними из компонентов архитектуры СППР.

Иногда различают «OLAP в узком смысле» – это системы, которые обеспечивают только выборку данных в различных разрезах, и «OLAP в широком смысле», или просто OLAP, включающей:

– 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 поставленных 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). Информация в MDD хранится не в виде индексированных записей в таблицах, а в форме логически упорядоченных массивов. Единой общепризнанной многомерной модели хранения данных не существует. В MDD отсутствует стандартизованный метод доступа к данным, и они могут отвечать требованиям специфической аналитической обработки данных.

Таблица 1.3

Сравнение OLTP и OLAP

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 целый 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;
- 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*
- *негосударственные*
  - частные
  - групповые
  - личные.

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

1.4. Уровни моделей и этапы проектирования БД

1.4.1. 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.

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 мы проектируем базу 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, 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 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 – ***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 отношение к нему.

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 будем называть ***инфологической*** (infological) ***моделью предметной области*** (ИЛМ)[1], 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.

1.4.2. 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 модель после этого могут строиться в любой последовательности по отношению друг к другу, в том числе и параллельно.

На рис. 1.12 изображена взаимосвязь этапов проектирования БД. Как видно из рисунка, при проектировании БД возможен возврат на предыдущие уровни. При этом возможны два вида возвратов: первый тип обусловлен необходимостью пересмотра результата проектирования (например, для улучшения полученных характеристик, «обхода»

[1] В предыдущих изданиях учебников С.М. Диго для этого уровня моделирования использовался термин «инфологическое». Последнее время в литературных источниках этот термин используется редко. В данном учебном пособии термины «инфологическое» и «концептуальное» моделирование используются как синонимы. (См.: Базы данных: проектирование и использование. – М.: Финансы и статистика, 2005; Проектирование баз данных. – М.: Финансы и статистика, 1988).

ограничений и т.п.), второй тип вызван необходимостью уточнения предыдущей модели (обычно – инфологической) с целью получения дополнительной информации для проектирования или при выявлении противоречий в модели.

Рис. 1.12. **Взаимосвязь этапов проектирования БД**

На рис. 1.13 и 1.14 изображены укрупненные технологические сети проектирования для этапов даталогического и физического проектирования[1]. Как видно из этих рисунков, результат предыдущего этапа проектирования используется на входе следующего этапа.

[1] 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.

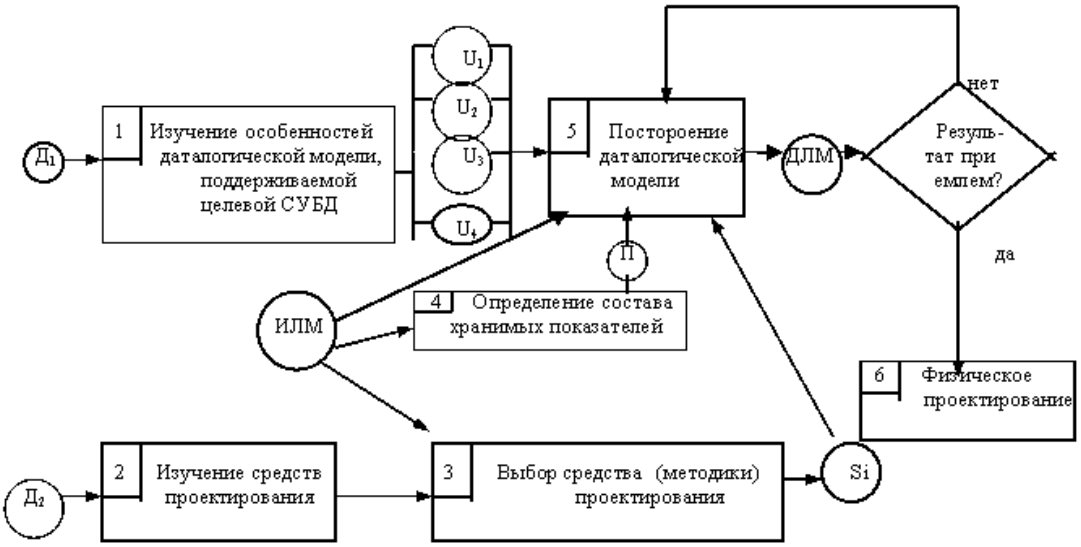


Рис. 1.13. **Технологическая сеть проектирования для этапа даталогического проектирования:**

$Д_1$ – 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; ДЛМ – даталогическая модель; ИЛМ – инфологическая модель; П – перечень хранимых показателей; Si – выбранное средство проектирования.

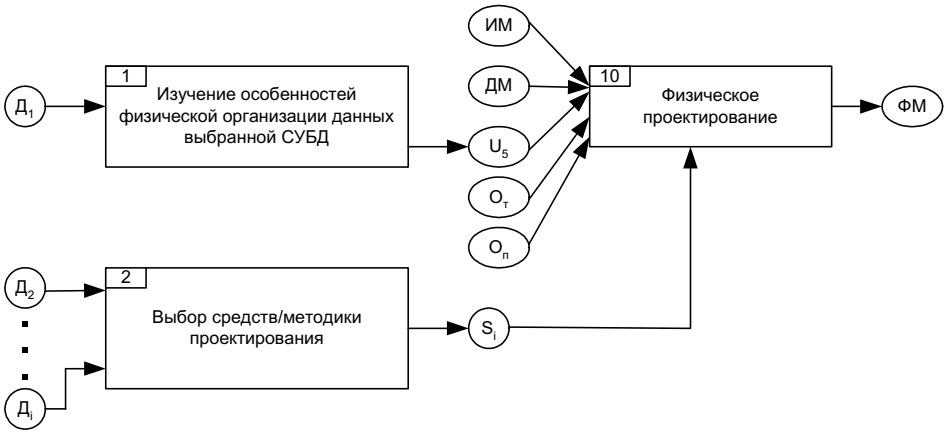


Рис. 1.14. **Технологическая сеть проектирования для этапа физического проектирования:**

$О_т$ – ограничения на используемые технические средства; $О_п$ – ограничения со стороны пользователей/процессов.

1.4.3. Факторы, влияющие 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 БнД. Для того чтобы было понятно, какая информация должна фиксироваться при описании предметной области, перечислим основные из факторов, оказывающих влияние на проектирование структуры БД:

1. Специфика предметной области:
   1.1. особенности отображаемых объектов, характер связи между объектами предметной области;
   1.2. «размер» системы (объем хранимых данных).
2. Особенности требуемой обработки информации:
   2.1. характеристика запросов (критерий поиска, частота запроса; состав реквизитов, выдаваемых в ответ, упорядоченность ответа, частота совместного использования реквизитов и т.п.);
   2.2. требования к защите информации;
   2.3. 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:
   3.1. важность/статус, приоритеты;
   3.2. число пользователей;
   3.3. распределение функций между пользователями; степень пересечения информационных потребностей пользователей;
   3.4. приоритеты пользователей в оценке значимости факторов, влияющих на проектирование БД;
   3.5. Технология обработки данных;
   3.6. возможность/необходимость работы в распределенной среде, в том числе необходимость поддерживать связь с «мобильными» компьютерами;
   3.7. «доступные» технологии обработки данных.
4. Состояние существующей системы обработки информации:
   4.1. наличие существующей автоматизированной системы обработки информации;
   4.2. объем имеющихся «наработок»;
   4.3. наличие технических и программных средств, их состояние;
   4.4. соотношение объемов «существующей» и «новой» частей проектируемой системы;
   4.5. затраты на перевод имеющейся системы на новую основу.
5. Возможности, предоставляемые используемыми (выбранными для реализации проекта) техническими и программными средствами:
   5.1. поддерживаемые структуры данных; ограничения, накладываемые программным обеспечением;
   5.2. ограничения по объему памяти;
   5.3. быстродействие технических средств;
   5.4. «производительность» программного обеспечения;
   5.5. особенности языков манипулирования данными.
6. Трудоемкость проектирования.

7. Финансовые возможности.
8. Квалификация кадров:
   8.1. разработчиков;
   8.2. пользователей.
9. Используемые методики проектирования:
   9.1. наличие средств автоматизации проектирования;
   9.2. используемый алгоритм проектирования.
10. Субъективные факторы:
   10.1. мода;
   10.2. привычки и предпочтения.

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

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 четвертого поколения.
12. Перечислите компоненты языка четвертого поколения.
13. Приведите примеры процедурных и непроцедурных языков. В чем основные отличия между языками этих классов?
14. Назовите основные отличительные особенности банков данных.
15. Какие технические средства необходимы для реализации банка данных?
16. Какие типы ЭВМ чаще всего используются для реализации банков данных?
17. Перечислите основные признаки классификации банков данных.
18. В чем разница между системами со структурированными и неструктурированными базами данных?
19. Охарактеризуйте основные классы СУБД.
20. СУБД каких классов являются в настоящее время наиболее распространенными?
21. Каковы основные тенденции развития СУБД в настоящее время?
22. Сравните системы типа OLTP и OLAP.
23. Перечислите основные отличия корпоративных и настольных СУБД.
24. Сравните локальные, интегрированные и распределенные БД.
25. Перечислите этапы проектирования баз данных.
26. Что называется схемой, подсхемой и внешней схемой?
27. Какую роль выполняет подсхема? Какие преимущества дает ее использование?
28. Что называется словарем данных?
29. Охарактеризуйте взаимодействие компонентов БнД при работе с системой.

30. Что называется инфологической моделью?
31. Является ли инфологическое моделирование этапом, присущим только проектированию баз данных?
32. Какая информация является исходной для построения концептуальной модели?
33. Кто должен создавать концептуальную модель и почему?
34. Какие требования предъявляются к инфологической модели?
35. Что называется даталогической моделью?
36. Какая информация является исходной для построения даталогической модели?
37. Какие вопросы решаются на стадии даталогического моделирования?
38. Изобразите технологическую сеть проектирования для стадии даталогического моделирования.
39. Что называется физической моделью?
40. Какая информация является исходной для построения физической модели?
41. Какие вопросы решаются на стадии физического моделирования?
42. Какие факторы влияют на проектирование БД?
43. Перечислите основные категории пользователей банков данных.
44. Кого называют конечными пользователями?
45. Кого называют администраторами банка данных?
46. Перечислите основные функции администратора банка данных.
47. В каком порядке должны выполняться этапы проектирования БнД?

Глава 2.

Концептуальное проектирование

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 в данной главе изложена некоторая базовая модель, с которой производятся сравнения других систем.

Так как в настоящее время CASE-систем достаточно много, то неизвестно, с какой именно из них проектировщику придется столкнуться на практике. Поэтому в данной главе даны некоторые критерии, по которым следует сравнивать CASE-системы, и приведены обобщенные рекомендации по построению ER-моделей в зависимости от доступных изобразительных средств и алгоритмов проектирования логической структуры базы данных.

В качестве примера рассмотрен процесс концептуального моделирования в среде AllFusion ERWin Data Modeler.

2.1. Общие сведения о моделировании предметной области

2.1.1. Уточнение понятия концептуальной модели

В базе данных отображается некоторая часть реального мира. Естественно, что полнота ее описания будет зависеть от целей создаваемой информационной системы. Часть реального мира, представляющая интерес для данного исследования, называется ***предметной областью*** (ПО). 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 (КМ),*** или ***инфологической (ИЛМ), моделью.*** Предметные области могут быть различными, и для их моделирования могут потребоваться специфические средства, соответствующие особенностям этих областей. В данном учебном пособии рассматриваются в основном 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 развития предметной области, при проектировании баз данных и программного обеспечения и др. Используемые средства и методы моделирования при этом будут различаться. В данном учебном пособии рассмотрены те средствах и методы моделирования, которые находят 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. Так как в настоящее время основную часть баз данных представляют структурированные базы данных, то основное внимание будет уделено проектированию именно таких систем.

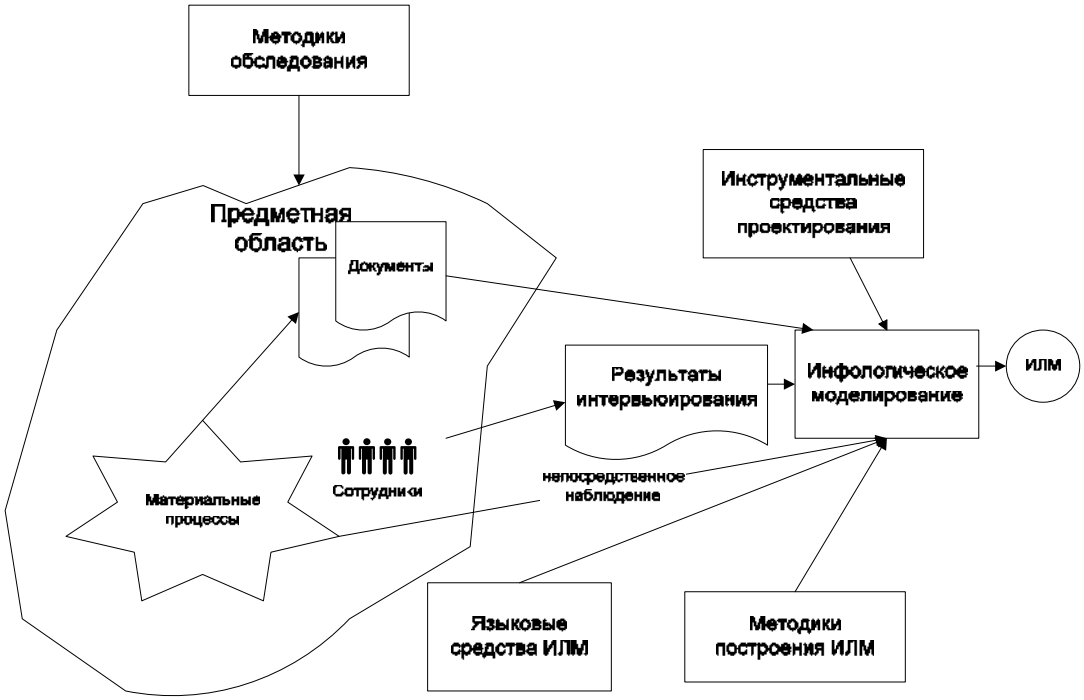


Рис. 2.1. **Стадия инфологического моделирования – исходная и результативная информация**

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 как «вручную», так и с использованием автоматизированных средств проектирования. Средства автоматизации проектирования отличаются как совокупностью 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.

2.1.2. Основные компоненты концептуальной модели

Основными компонентами концептуальной модели ПО являются:

- 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.

Далее мы 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);
- 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; ИЛМ 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 (в связи с большой размерностью реальных инфологических моделей должна обеспечиваться возможность ее композиции и декомпозиции).

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

- быть вычисляемым, то есть 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 обеспечить, то необходимо, чтобы первые могли хотя бы проверить сделанное другим специалистом описание, чтобы убедиться, 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 (и, 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) и диалектов SQL конкретных СУБД;
- при смене используемой СУБД не надо 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» (то есть получения ER-диаграммы по имеющимся описаниям данных) позволяет использовать существовавшие ранее наработки для «реинжиниринга» системы;
- указание связи объектов в ER-модели и соответствующая миграция ключа при преобразовании этой модели в целевую позволяет не только задавать контроль целостности связи при ведении БД, но и автоматически обеспечивает 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. Автор данного учебного пособия предлагает собственную методику (бу-

дем называть её базовой) – она и будет рассматриваться ниже. С некоторыми другими методиками можно познакомиться в п. 2.3, а также по литературным источникам[1] или по документации по CASE-средствам.

2.2.1. Понятия «объект» и «класс объектов»

В предметной области имеется множество разнообразных объектов. 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 другие.

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 – им является грамматический оборот существительного (существительное, у которого могут быть определения 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-модели 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 мы 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 в реальной жизни так бывает не всегда (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***. 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.

[1] Вендров А.М. Case-технологии. Современные методы и средства проектирования информационных систем. – М.: Финансы и статистика, 1988; Джексон Г. Проектирование реляционных баз данных для использования с микроЭВМ / Пер. с англ. – М.: Мир, 1991; Когаловский М.Р. Энциклопедия технологий баз данных. – М.: Финансы и статистика, 2002.

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 рассмотрении объект может считаться простым, а в другом этот же объект может рассматриваться как сложный. Так, например, объект АУДИТОРИЯ, в случае 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 родо-видовых связей означает классификацию объектов предметной области по тем или иным признакам. Естественно, 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. Имя агрегированного объекта обычно является отглагольным существительным.

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. 2.2). Если у объекта несколько имен, то для каждого из них выделяется отдельный «сектор» в этом прямоугольнике.

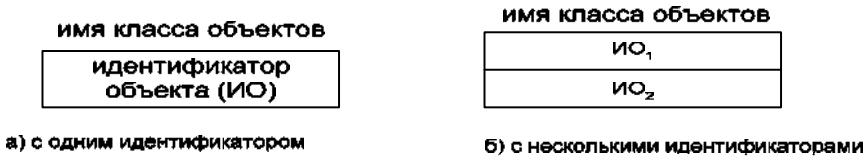


Рис. 2.2. **Изображение объекта**

Если какое-либо имя объекта не является уникальным, будем использовать букву «н» рядом с таким именем (рис. 2.3).

Рис. 2.3. **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, то есть участок имеет составной идентификатор НОМЕР_ЦЕХА*НОМЕР_УЧАСТКА. Назовем подобные объекты (в нашем примере это объект «УЧАСТОК») ***зависимыми от идентификации сущностями***. Для отображения таких ситуаций будем перечеркивать линию, соединяющую соответствующие объекты, около конца, прилегающего к зависимому объекту (рис. 2.4, 2.5).

Рис. 2.4. **Изображение 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 несколько имен.

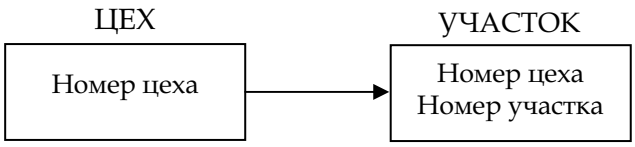


Рис. 2.5. **Пример изображения зависимой по идентификации сущности**

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 надо изобразить набор свойств, фиксируемых для объектов каждого из представленных в модели классов. Для обозначения свойств будем использовать прямоугольник, изображенный пунктирной линией.

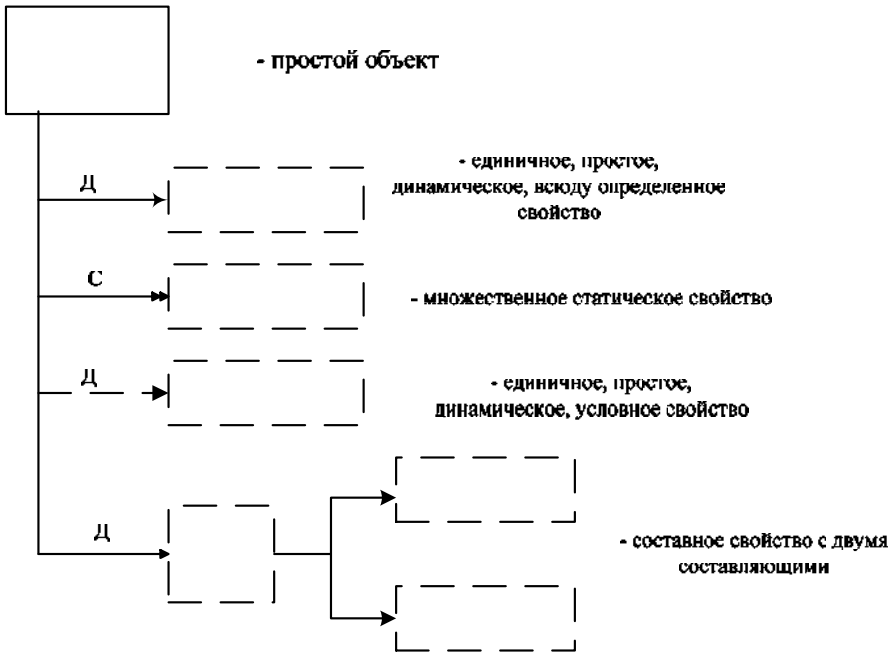


Рис. 2.6. **Изображение объекта и его свойств (условные обозначения)**

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. Назовем такие свойства ***единичными*** Например, каждый человек может иметь только одну ДАТУ_РОЖДЕНИЯ или СТАЖ_ РАБОТЫ. Для других свойств возможно существование одновременно нескольких их 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).

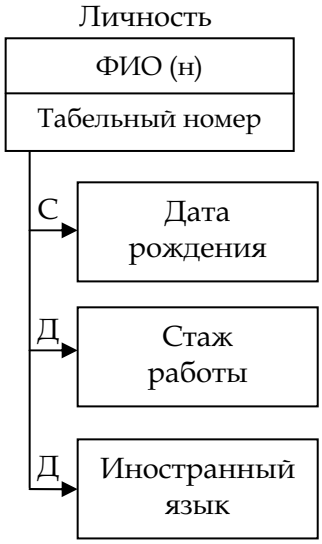


Рис. 2.7. **Пример изображения единичных и множественных динамических и статических свойств**

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 (см. рис. 2.7).

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 из 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 в случае если свойство определено для всех экземпляров объектов данного класса – сплошную (рис. 2.8).

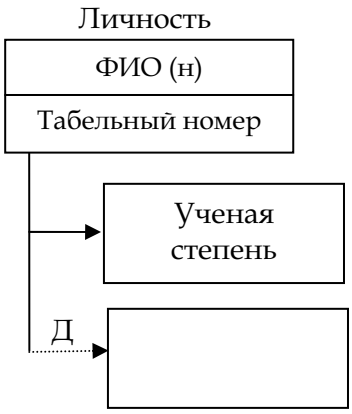


Рис. 2.8. **Пример изображения условного свойства**

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. 2.9а, а если внутривузовское совместительство не разрешено – как на рис. 2.9б.

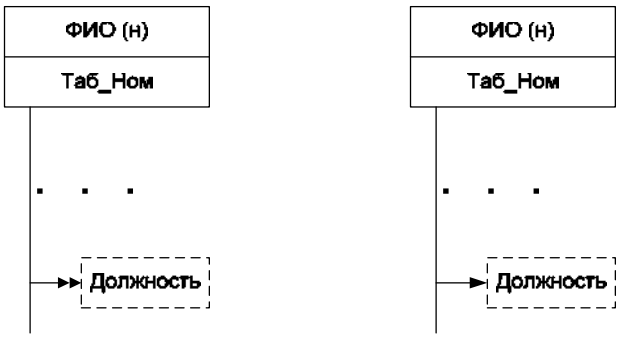


Рис. 2.9. **Варианты изображения (в зависимости от особенностей предметной области)**

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 (рис. 2.10).

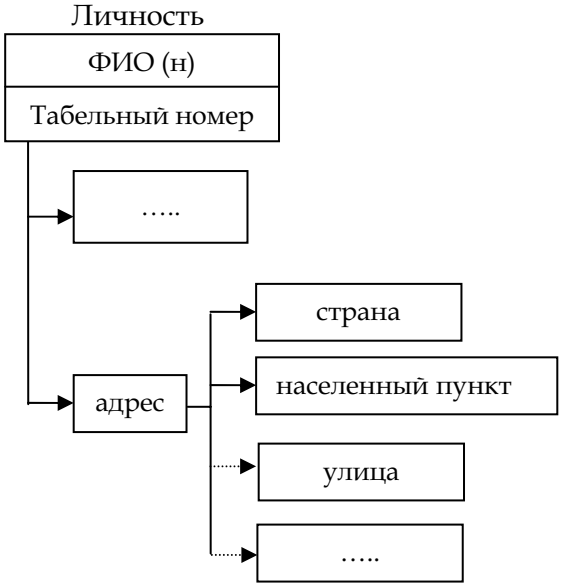


Рис. 2.10. **Пример изображения составного свойства**

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 это невозможно в целевой СУБД, то:

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

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 ничего, соответствующее 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, такие характеристики рекомендуется отображать не непосредственно в модели, а в каталоге реквизитов.

Таблица 2.1

Сводная таблица характеристик свойств объекта

Характеристика	Что отражает	Возможные значения	Пример*
Назначение	что определяет	• идентификатор (обозначает, называет объект)	ФИО
		• качественная характеристика	пол
		• количественная характеристика	вес
		• дата/время совершения события	дата рождения
		• изображение	фотография
Изменяемость значения свойства	может ли меняться в течение жизненного цикла объекта	• динамическое	образование
		• статическое	дата рождения
Обязательность	может ли отсутствовать данное свойство у объекта	• обязательное	дата рождения
		• необязательное	ученая степень
Элементарность	возможность разбиения на составляющие элементы	• простое	пол
		• составное	адрес
Множественность	возможность наличия нескольких значений для данного свойства у одного объекта	• единичное	дата рождения
		• множественное	номера телефонов
Форма отображения в знаковой системе		• символьная	адрес
		• числовая	вес
		• дата	дата рождения
		• время	время прихода на работу
		• изображение	фотография сотрудника
		• логическое	военнообязанный (да/нет)
Способ получения		• датчики/счетчики	кардиограмма
		• из внешней среды	рекомендации с прежней работы
		• производная информация (вычисляемая)	стаж работы
		•	

* примеры приведены для класса объектов ЛИЧНОСТЬ.

Понятия «объект» и «свойство» являются относительным. 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 производится. 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.

В общем случае можно дать следующие рекомендации по поводу того, что следует выделять в качестве самостоятельного объекта в ER-модели. 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.

В экономических организационных системах большая часть информации отражается в символьном виде. При этом различают реквизиты-признаки, отражающие 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 из реквизитов 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. 2.11). Расчетные формулы, а также алгоритмы вычислений в том или ином виде должны быть представлены в ИЛМ.

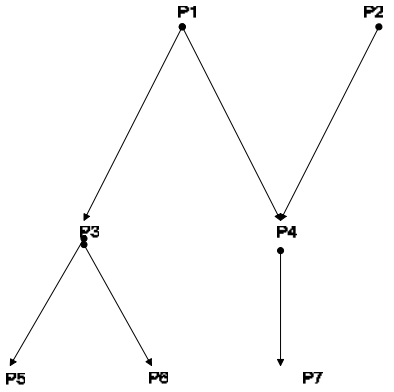


Рис. 2.11. **Граф взаимосвязи показателей**

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 из сотрудников, 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 специальное обозначение; в нашем случае это прямоугольник, очерченной двойной линией (рис. 2.12).

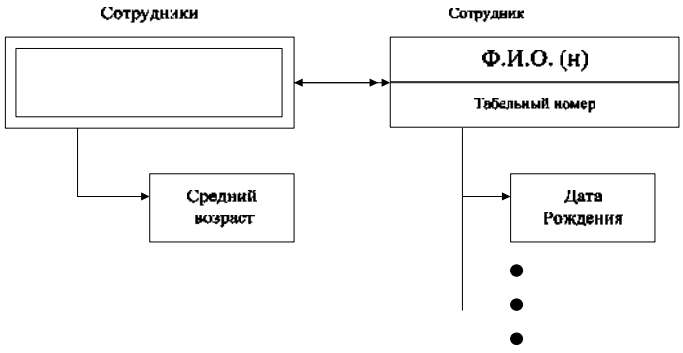


Рис. 2.12. **Отображение интегральных характеристик класса объектов**

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 постоянны-

ми, то для соответствующих показателей желательно фиксировать максимальное, минимальное и среднее число объектов в классе.

Таблица 2.2

Описание классов объектов (фрагмент)

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	-	15000		
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

Количественные характеристики классов объектов используются не только для определения объема памяти, занимаемого БД, но также 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, в приведенном выше примере число объектов в классе СТУДЕНТ является постоянным (5000), но ежегодно 1/5 часть студентов оканчивает институт и столько же новых студентов поступает. Это означает, что ежегодно добавляется 20% от общего объема данных 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.

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 между сущностями одного класса. Например, связь «Быть руководителем» устанавливается между разными экземплярами объектов одного класса «СОТРУДНИК».

Различают связи типа «*один к одному*» (1:1), «*один ко многим*» (1:M), «*многие к одному*» (M:1) и «*многие ко многим*» (M:M). Иногда эти типы связей называются степенью связи (или кардинальностью). На рис. 2.13 изображены условные обозначения для каждого из этих видов связи.

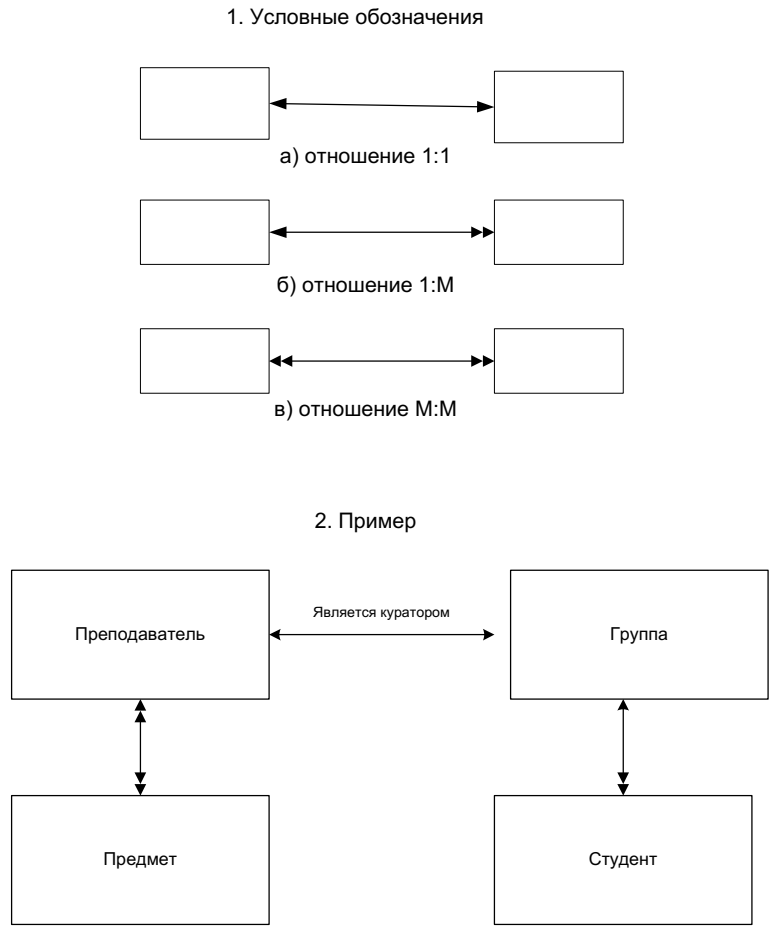


Рис. 2.13. **Виды связей между объектами**

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

Возможны случаи, когда между парой объектов объявлено несколько связей (рис. 2.14). В этом случае связи следует именовать (указание роли).

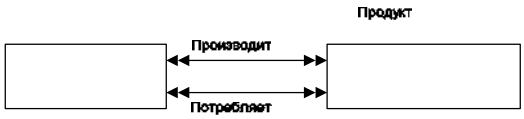


Рис. 2.14. **Пример объявления двух связей между парой объектов**

Кроме степени связи в ER-модели, для характеристики связи между разными сущностями надо 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. Линия, соединяющая объекты, может быть полностью сплошной, полностью пунктирной либо наполовину сплошной, наполовину пунктирной (рис. 2.15).

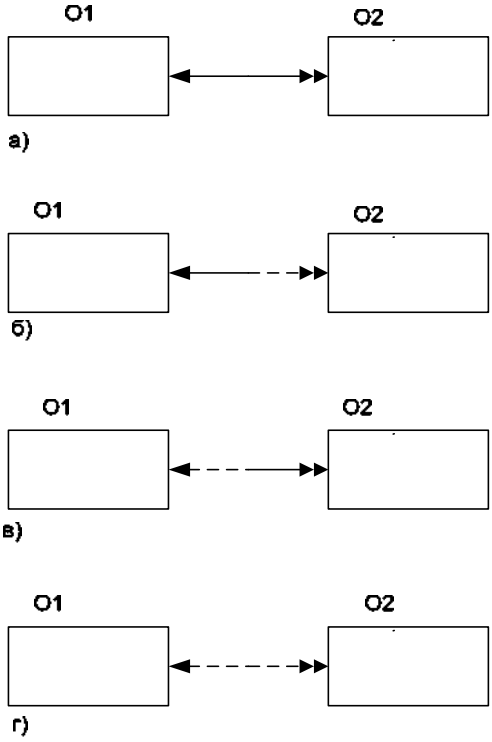


Рис. 2.15. **Варианты классов членства**

Поясним суть понятия класса членства на конкретных примерах. Для графической иллюстрации используем так называемые диаграммы ER-экземпляров (вместо диаграмм ER-типов, 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. 2.16а), а диаграмма ER-типов – как на рис. 2.16б). 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 членства – необязательный с обоих концов (линия – полностью пунктирная).

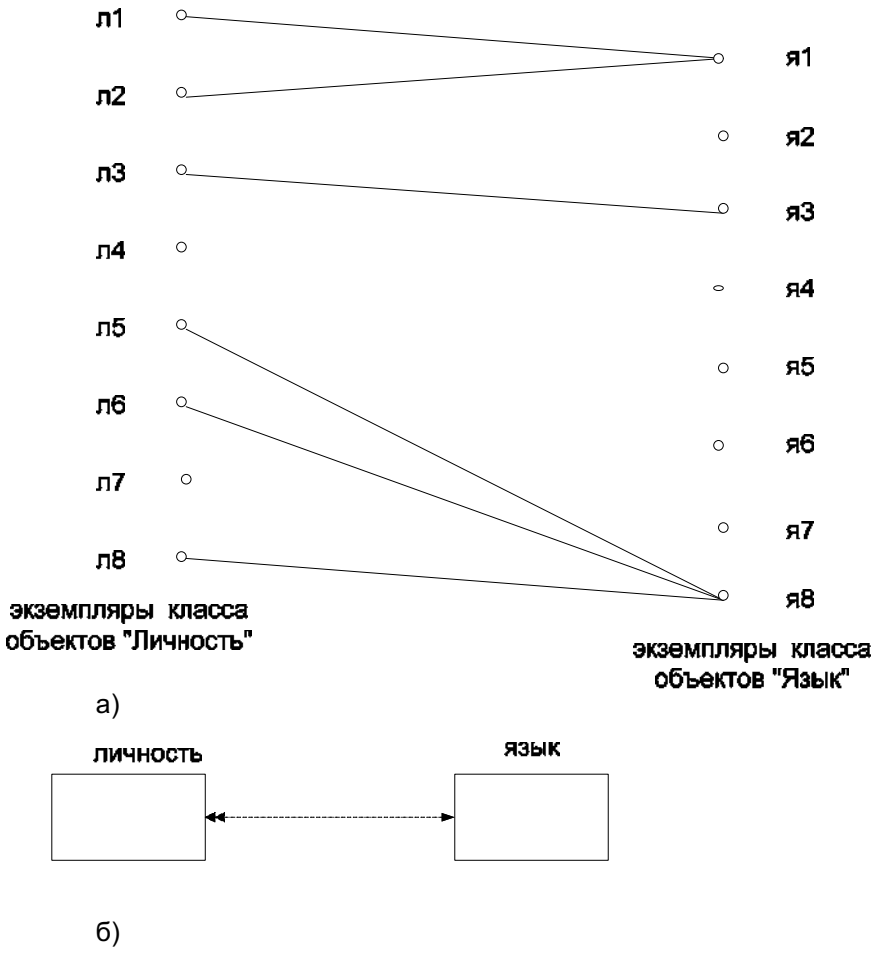


Рис. 2.16. **Диаграмма ER-экземпляров (а) и ER-типов (б)**
(вариант 1 – необязательное членство с обоих концов связи)

Другой вариант: 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 абитуриентов, остается актуальным и в этой ситуации. В этом случае диаграмма ER-экземпляров будет иметь вид, изображенный на рис. 2.17а, а диаграмма ER-типов – как на рис. 2.17б.

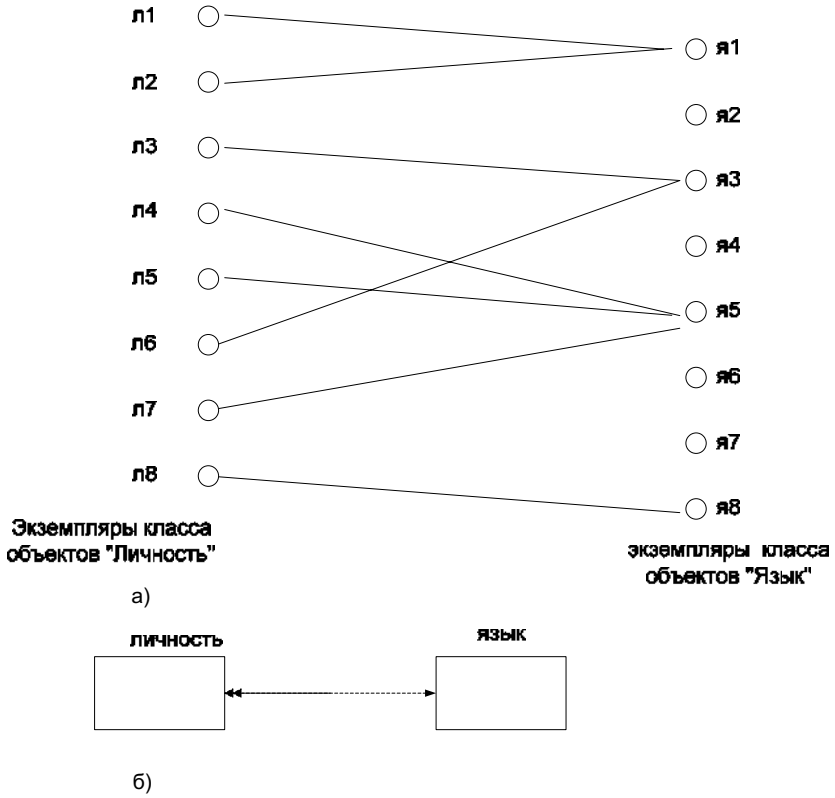


Рис. 2.17. **Диаграмма ER-экземпляров (а) и ER-типов (б) (вариант 2 – связь 1:М и необязательное членство с одного конца связи)**

Как в первом, так и во втором из рассмотренных случаев 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 является обязательным, а для сущности ЯЗЫК_ИНОСТРАННЫЙ – необязательной. На диаграмме это будет отображено пунктирной линией, прилегающей к объекту ЯЗЫК_ИНОСТРАННЫЙ, и сплошной линией, прилегающей к объекту ЛИЧНОСТЬ.

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 членства объекта СТУДЕНТ в связи с объектом «ГРУППА» будет необязательным, в противном случае он будет обязательным.

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. 2.18).

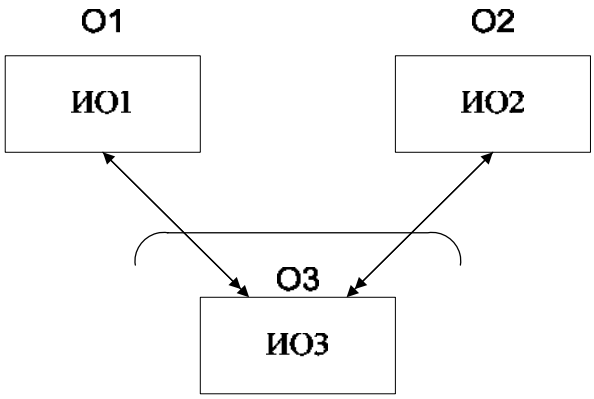


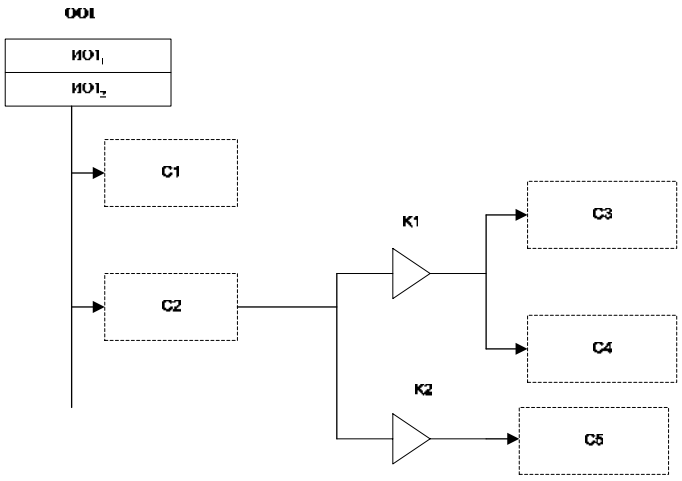
Рис. 2.18. **Изображение альтернативной связи**

2.2.8. Сложные объекты

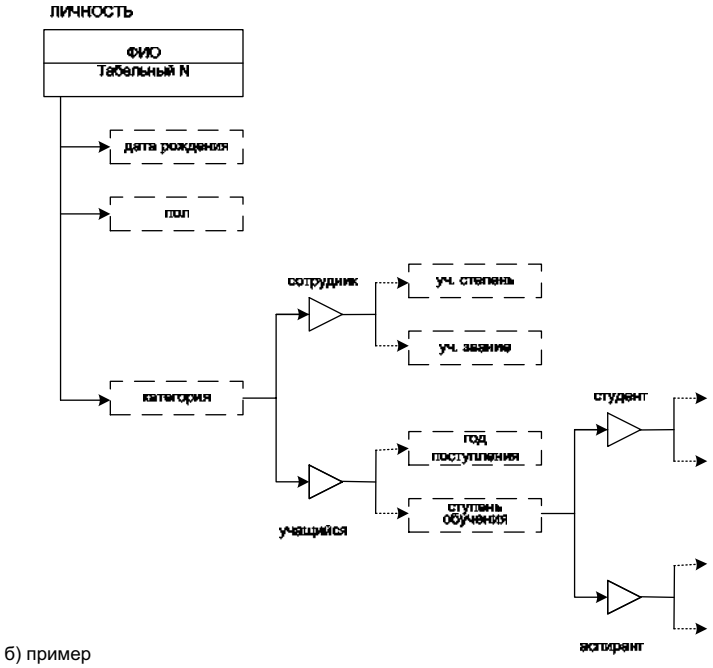
Различают три разновидности сложных объектов: обобщенные, агрегированные и составные.

Обобщенный объект представляет собой объект, в котором явным образом выделены подклассы. Разбиение класса на подклассы осуществляется по определённому признаку (свойству). Свойство, по которому производится разбиение класса на подклассы, называется ***дискриминатором***. Например, подклассы ВОЕННООБЯЗАННЫЕ и НЕВОЕННООБЯЗАННЫЕ выделяются в зависимости от значения свойства «отношение к воинской обязанности»; подклассы ДОВУЗ, СТУДЕНТЫ, АСПИРАНТЫ, ДОКТОРАНТЫ, выделяются в зависимости от значения свойства «вид обучения».

Для обозначения подкласса в схеме будем использовать треугольник, который связан с обозначением свойства, по которому производится разбиение на подклассы, и к которому присоединены обозначения свойств, присущих данному подклассу. Широкая часть треугольника направлена в сторону «родового» объекта, острый угол – в сторону «видового» (рис. 2.19а).



а) условные обозначения



б) пример

Рис. 2.19. **Изображение обобщенного объекта**

На рис. 2.19б изображен фрагмент инфологической модели, отражающий обобщенный объект ЛИЧНОСТЬ для высшего учебного заведения. Для него выделено несколько категорий объектов: ПРЕПОДАВАТЕЛЬ, СТУДЕНТ, АСПИРАНТ.

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.

Подклассы в совокупности могут составлять исходный класс (полный класс), а могут представлять лишь её часть (неполный класс). Если при описании предметной области возникает необходимость отобразить эту информацию, то для полного класса будем

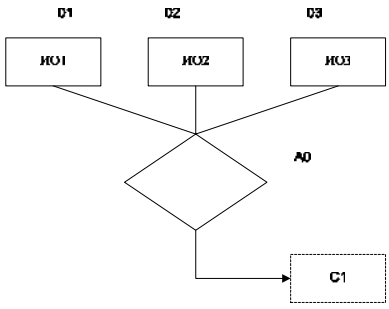
использовать двойную линию, перечеркивающую линию, идущую от дискриминатора; если класс неполный – одинарную линию, перечеркивающую линию, идущую от дискриминатора.

Подкласс, как и класс, является совокупностью однотипных объектов. Отображать 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.

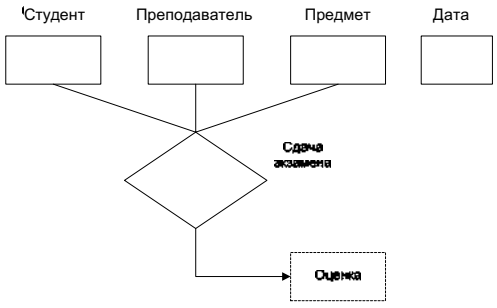
При использовании обобщенного объекта связи между объектами могут идти как к знаку всего обобщенного объекта, если объекты всех подклассов участвуют в данной связи, так и к знаку отдельного подкласса, если связь относится только к данному подклассу.

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

Агрегированные объекты 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. Этот ромб связывается линиями с условными обозначениями тех объектов, которые участвуют в изображаемом процессе. Свойства агрегированного объекта изображаются так же, как и для простого объекта (рис. 2.20а).



а) условные обозначения



б) пример

Рис. 2.20. **Изображение агрегированного объекта**

В качестве примера агрегированного объекта из рассматриваемой предметной области «Учебный процесс» изобразим СДАЧУ ЭКЗАМЕНА (рис. 2.20б).

Составной объект отражает связь типа «целое – часть». Для отображения составных объектов в ER-модели обычно не используются 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 производиться только для хранимых показателей.

Как отмечалось выше, понятия «объект» и «свойство» являются относительным. В общем случае можно дать следующие рекомендации по поводу того, что следует выделять в качестве самостоятельного объекта в ER-модели. 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.

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.

При изображении предметной области надо стремиться отобразить информацию как можно более детально, так как это в дальнейшем 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 как обобщенный объект, приходится выбирать между двумя крайними вариантами: надо ли простой объект представить как обобщенный и надо ли два или несколько самостоятельных объектов «объединить» в обобщенный объект.

Обобщенный объект следует вводить в модель в том случае, когда надо 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 поля 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 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. 2.21 изображены фрагменты ER-модели, отображающие факт знания сотрудником иностранных языков. Каждый из изображенных вариантов при всем их различии являются правильными.

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.

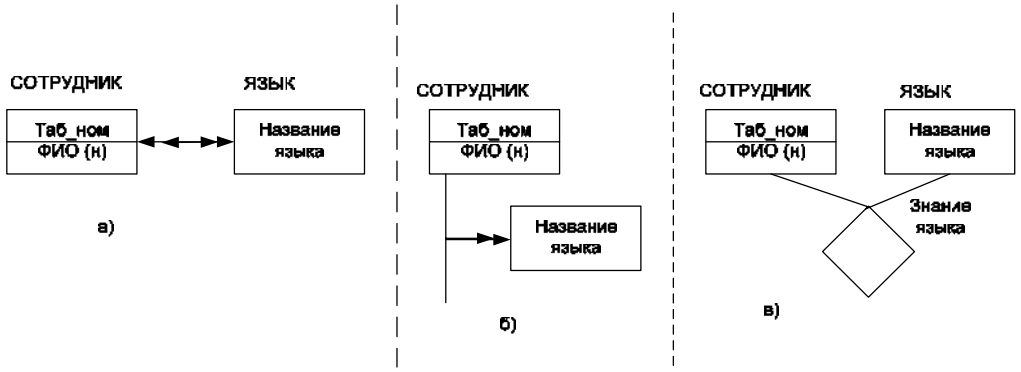


Рис. 2. 21. **Варианты изображения связи «СОТРУДНИК – ЯЗЫК»**

В заключение данного раздела хотелось бы обратить внимание на некоторые наиболее часто допускаемые ошибки в процессе моделирования. Одной из таких ошибок является изображение взаимосвязанных свойств в виде самостоятельных, не связанных друг с другом. Так, на рис. 2.22а изображен неправильный вариант отображения информации о степени владения сотрудником тем или иным иностранным языком. Кроме правильного варианта, изображенного на рис. 2.22б, возможен вариант, аналогичный изображенному на рис. 2.21б, где СТЕПЕНЬ_ВЛАДЕНИЯ будет свойством агрегированного объекта ЗНАНИЕ_ЯЗЫКА.

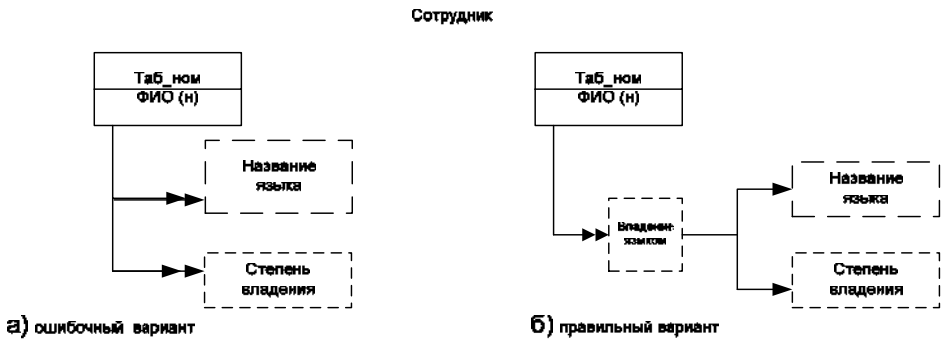


Рис. 2.22. **Изображение информации о владении иностранными языками в базовой ER-модели**

Следует помнить, что нельзя устанавливать связь между свойством одного объекта и другим объектом или свойством. Нельзя непосредственно связывать между собой агрегированные объекты.

Глава 3.

Даталогическое проектирование

Подходы к проектированию логической структуры БД существенно зависят от типа модели данных. В данной главе будут рассмотрены вопросы даталогического проектирования применительно к структурированным моделям данных.

3.1. Общие сведения о даталогическом проектировании

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.

Любая СУБД оперирует с допустимыми для нее логическими единицами данных, а также допускает 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 является проектированием логической структуры 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, является отображением реальной предметной области. В связи с этим специфика отображаемой предметной области, отраженная в инфологической модели, оказывает самое непосредственное влияние на выбор проектных решений при проектировании базы данных.

Каждый тип модели данных и каждая разновидность модели, поддерживаемая конкретной СУБД, имеют 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 инфологической 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 осуществляется 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 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 М:М между элементами. В этом случае в даталогическую модель вводится дополнительный вспомогательный элемент, отображающий эту связь (таким образом отношение М:М как бы разбивается на два отношения 1:М 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. 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.

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, напротив, обобщенный объект как единое целое не отображается в структуре базы данных.

Другие способы отображения обобщенных объектов 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.

Как отмечалось выше, характер обработки информации также 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.

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 хранении 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 принимаются во внимание несколько факторов.

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 получил 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 стипендии», 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 представляет собой некоторую проблему. При правильной организации файлов вычисление средней величины во время исполнения запроса не представляет проблемы ни с точки зрения времени обработки, ни с точки зрения задания запроса. Поэтому хранить средний балл не следует.

3.2. Критерии оценки БД

Для оценки проектируемой/спроектированной БД может быть использовано множество критериев. Значимость этих критериев в свою очередь будет 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 (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 показателям. Необходима комплексная оценка проекта по всей совокупности критериев.

Критерии могут быть как количественными, так и качественными.

Перечислим основные критерии, используемые при оценке БД:

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, так как некоторые несоответствия очень трудно выявляются. 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.

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.

Адаптируемость – приспособление к изменяющимся условиям. 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. Обеспе-

чение 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. Первый вариант предполагает, что создается таблица, в строках которой 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 не существенно, а для оценки БД по другим критериям будет иметь значение, сколько таблиц будет 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	….

Во втором варианте 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 первый вариант, то при каждом изменении в учебном плане надо будет менять структуру базы данных. Во втором случае никакие изменения структуры БД не потребуются. Более того, при создании БД по второму варианту вообще не надо 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. *Она* 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.

Приведем другой пример, показывающий влияние выбранного типа данных на устойчивость спроектированной БД. Предположим, что в предметной области для кодирования какой-либо номенклатуры используется цифровой код и в базе данных для со-

ответствующего 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 текстовый, 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.*

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 «*необходимое число изменений, которые необходимо провести при наступлении одного события в предметной области*».

Например, предположим, что мы имеем базу данных, содержащую сведения о сотрудниках учебного заведения, и фиксируем в ней некоторые биографические и прочие справочные данные, а также информацию 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.

Недостатком варианта 1 является то, что в случае 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.

Первое проектное решение – ненормализованная структура. Она плоха тем, что приводит к большому дублированию информации. Второй и третий вариант – оба представляют нормализованную структуру и отличаются тем, что в последнем варианте введен искусственный идентификатор. Именно третий вариант 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.

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.

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.

Сложность структуры БД. Речь может вестись 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 связывания, 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.

Поэтому критерий «сложность» никогда не рассматривается как самостоятельный.

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, а также часто ведет и к увеличению числа полей в БД, что увеличивает ее сложность.

Сложность последующей обработки. Оценить этот показатель достаточно трудно, так как его значение зависит как от предполагаемой обработки, так и от возможностей языка манипулирования данными конкретной СУБД. Тем не менее, для большинства СУБД справедливыми являются утверждения, что:

- 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 надо вывести фамилию и инициалы, то в случае раздельного хранения полей это также легко сделать, например, просто изменив шаблон вывода для полей «имя» и «отчество» на (X.)). Однако

хранение каждой из составляющих СЕИ (составная единица информации) в виде отдельных полей имеет и очевидные недостатки: база данных становится сложнее, затрачивается больше времени при создании файла на описании 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 вы в БД «КАДРЫ» включили поле «ДЕТИ», в котором в записи о сотруднике фиксируете сведения обо всех его детях, то задать запросы типа «Выделить всех сотрудников, имеющих больше 3-х 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, то 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.

Таким образом, при проектировании БД необходимо знать:
а) особенности языков манипулирования данными в целевой СУБД;
б) особенности предполагаемой обработки данных.

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 определении 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.

Скорость (время) обработки информации (время реакции на запрос). Значение данного критерия очень трудно достаточно точно оценить на стадии проектирования, так как 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 надо 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 это не учесть 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 вы выберете в качестве ключа поле, значение которого может изменяться, то у вас могут возникнуть проблемы при поддержании целостности БД.

Таким образом, при проектировании БД надо, с одной стороны, оценить предметную область с точки зрения ее изменчивости, а с другой стороны – 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 эту систему перенести без перепроектирования в машинную форму, то это повлечет за собой создание по меньшей мере столько 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, так как 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 структуру чрезвычайно тяжело. Например, чтобы знать, какие оценки у студента Х по физике, надо знать, в какой группе учится данный студент, какая/какие таблицы содержат 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. Т.е. такое решение неудачно по всем основным показателям: плохие адаптивные свойства, велика сложность структуры и обработки. Универсальность системы равна 0.

Если предложить другой вариант, а именно, создать отношение «УСПЕВАЕМОСТЬ» с полями «КОД_СТУДЕНТА», «ПРЕДМЕТ», «СЕМЕСТР», «ОЦЕНКА», то все указанные недостатки будут устранены.

Недостатком второго варианта по сравнению с первым будет бóльшая степень дублирования данных: в первом варианте ФИО/ КОД_СТУДЕНТА фиксируется только один раз как значение поля, а названия предметов вообще фиксируются только в описании структуры, а во втором варианте они будут повторены как значения соответствующих полей при фиксации каждого факта сдачи экзамена. Тем не менее предпочтение все равно должно быть отдано этому варианту.

3.3. Особенности даталогических моделей

В базах данных со структурированными моделями следует различать *внутризаписную* и *межзаписную* структуру. 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 случае иерархической внутризаписной структуры 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 мы имеем 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. В случае 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, либо возможно отсутствие каких-либо полей, либо допускается разное число экземпляров для повторяющихся компонентов.

Поле является наименьшим семантически значимым элементом записи. Основными характеристиками поля является 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.

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.

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.

В сетевых моделях, если на них 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 при помощи различных указателей.

В реляционной модели 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). Хотя 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, так как в этом случае многие проблемы (в частности, обеспечение проверки на уникальность ключа и соблюдение некоторых других ограничений целостности) возлагаются на поль-

зователя (проектировщика ИС). Вне зависимости от того, требует СУБД явного указания ключей при описании отношений или нет, проектировщик базы данных должен понимать, что́ является ключом каждого отношения.

В случае если СУБД поддерживает концепцию ключа, при наличии нескольких вероятных ключей один из них выбирается и описывается как *первичный ключ*, остальные – как альтернативные (обычно для этих целей используется либо уникальный индекс, либо описатель UNIQUE при определении таблицы).

Атрибут или группа атрибутов, которая в рассматриваемом отношении не является ключом, а в другом отношении ключом является, называется *внешним ключом*.

Если таблица содержит внешний ключ, то она: а) логически связана с таблицей, содержащей соответствующий первичный ключ; б) эта связь имеет характер «один-комногим» (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)*. Он представляет собой виртуальную таблицу, часто полученную в результате логического соединения нескольких связанных по значениям общих столбцов таблиц и, возможно, включающую 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.

Существует еще одна разновидность моделей данных – *системы, построенные на основе инвертированных файлов*. С точки зрения логической структуры БД эти 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 информации от собственно хранимых данных позволяет изменять связи, не изменяя при этом самих файлов.

3.4. Проектирование логической структуры реляционной базы данных

3.4.1. Вводные положения

Для реляционной базы данных проектирование логической структуры заключается в том, чтобы разбить всю информацию по файлам[1] (или в терминах реляционной мо-

[1] В некоторых СУБД каждой таблице ставится в соответствие файл базы данных, в других – вся база данных хранится в одном файле. В этом параграфе термины «файл», «таблица» и «отношение» используются как синонимы и не отражают способ физического хранения данных.

дели – по отношениям, таблицам), а также определить состав полей (в терминах реляционной теории – атрибутов) для каждого из этих файлов. 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.

Мы 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 аналогичный подход.

3.4.2. Алгоритм перехода от базовой ER-модели к схеме реляционной базы данных

Ниже описан 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 компонент концептуальной модели предметной области.

Рассмотрим рекомендации по переходу от базовой ER-модели к схеме реляционной базы данных для каждого из типов элементов ER-модели.

Отображение простых объектов

Для каждого простого объекта и его единичных свойств строится отношение, атрибутами которого являются идентификаторы объекта и реквизиты, соответствующие каждому из единичных свойств (рис. 3.1).

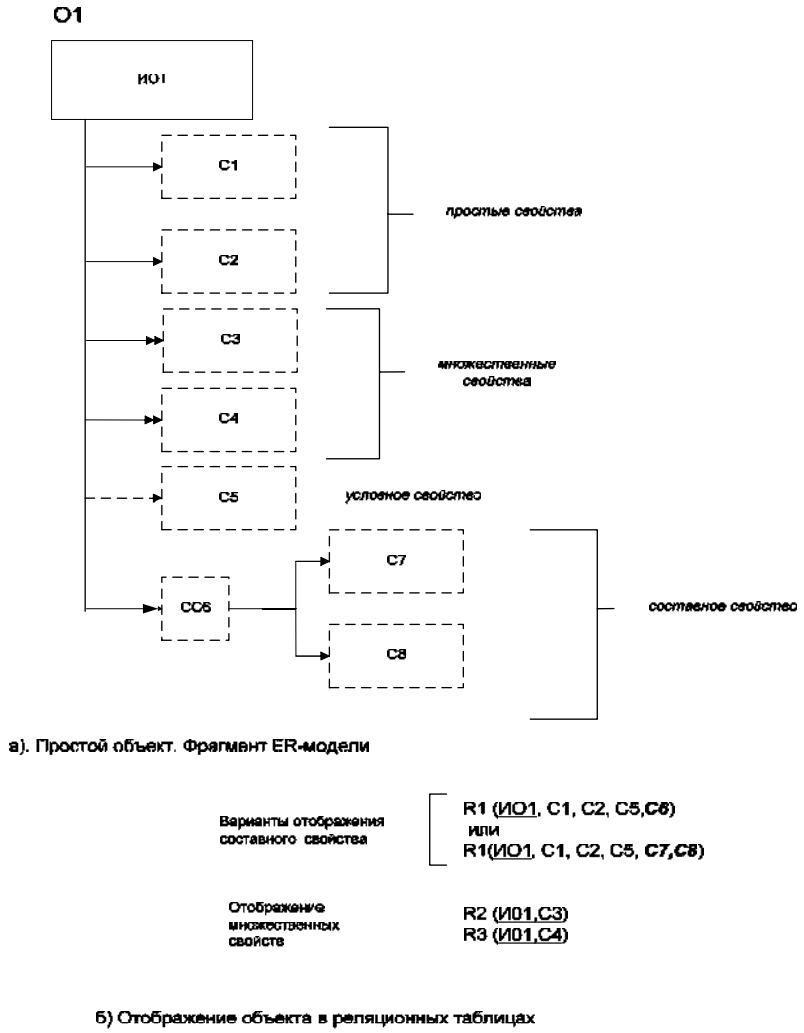


Рис. 3.1. **Отображение единичных, множественных и составных свойств**

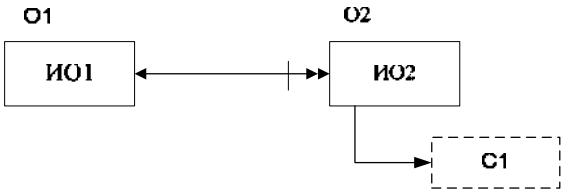
Любой из уникальных идентификаторов объекта является вероятным ключом полученного отношения. Если объект имеет несколько уникальных идентификаторов, необходимо один из них выбрать в качестве первичного ключа. Часто (но не обязательно) в качестве первичного ключа выбирается самый короткий из вероятных ключей. На решение вопроса о выборе первичного ключа (кроме длины ключа) влияют следующие факторы:

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. Так как 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 каче-

стве 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 составной, включающий идентификатор этого объекта и идентификатор «вышестоящего» объекта или, как говорят, идентификатор «основного» объекта «мигрирует» в таблицу, соответствующую зависимому объекту (рис. 3.2). 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 тех случаях, когда надо отображать связь этого «зависимого» объекта с другими.



а). Изображение зависимой по идентификации сущности. Фрагмент ER-модели

R2 (И01, ИО2, С1)

б) Отображение зависимой по идентификации сущности в реляционных таблицах

Рис. 3.2. **Отображение зависимой по идентификации сущности**

Обычно зависимый по идентификации объект и тот объект, от которого он зависит, связаны друг с другом отношением 1:М, и может сложиться впечатление, что перенесение ключа просто соответствует отображению этой связи. Однако это не совсем так. В случае если сущность независима по идентификации, то связь 1:М можно отображать в структуре БД как путем переноса ключа связанного объекта в таблицу, соответствующую подчиненному объекту (т.е. объекту, стоящему со стороны М), так и другими способами, а ключом таблицы, соответствующей подчиненному объекту, будет являться только идентификатор самого этого объекта. В случае зависимого по идентификации объекта связь 1:М 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 использования 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 использовать при создании таблицы, соответствующей агрегированному объекту, то он вряд ли 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.), поэтому для отображения этой информации может потребоваться отдельная таблица и код «сдачи экзамена» может оказаться полезным).

Другими словами, при создании таблицы в каждом конкретном случае надо решать, 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.

Если у объекта имеются множественные свойства, то каждому из таких свойств ставится в соответствие отдельное отношение, полями которого будут идентификатор объекта (если у объекта несколько идентификаторов, то тот, который выбран в качестве первичного ключа) и поле, соответствующее множественному полю. Ключ этого отношения будет составным, включающим оба эти атрибута (см. рис. 3.1).

Приведенное выше решение является универсальным. В отдельных случаях могут быть приняты и другие решения. Так, если число экземпляров множественного свойства у каждого из объектов 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 надо.

Если объект обладает условными свойствами, то при отображении их в реляционную модель возможны следующие варианты:

1) если многие из объектов 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 сложности ее обработки) и применяется сравнительно редко.

На рис. 3.1 использован вариант «а». Если бы было выбрано второе из обсуждаемых решений, то из отношения R1 атрибут С5 следовало бы исключить и создать дополнительно новое отношение R4 (ИО1, С5).

Если объект имеет составное свойство, то возможны два способа его отображения в БД:

1) всему составному свойству ставится в соответствие одно поле;

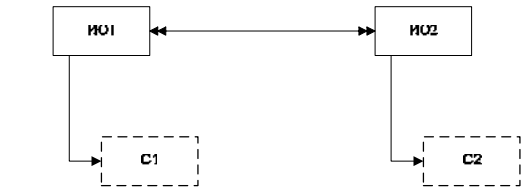
2) каждому из составляющих элементов составного свойства ставится в соответствие отдельное поле.

Выбор варианта будет зависеть, в основном, от характера преимущественной обработки этой информации: так как 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 использовать вариант 1, в противном случае – вариант 2.

Связи между объектами также должны отражаться в структуре БД. Универсальным способом отображения связи между объектами является введение вспомогательного связующего файла, содержащего идентификаторы связанных объектов. Ключ этого отношения будет составным. Такое решение является практически единственно приемлемым при наличии связи М:М между объектами. Дополнительными доводами в пользу такого решения является также 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.

Если между объектами предметной области имеется связь M:M, то для хранения такой информации потребуется три отношения: по одному для каждой сущности и одно дополнительное – для отображения связи между ними. Последнее отношение будет содержать идентификаторы связанных объектов (рис. 3.3). Ключ этого отношения будет составным.



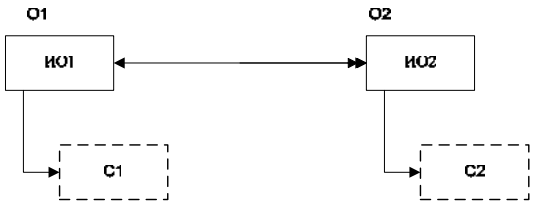
а). Фрагмент ER-модели

R1(ИО1,С1)
R2(ИО2, С2)
RCB(ИО2, ИО1)

б). отображение в реляционную модель

Рис. 3.3. **Отображение связи M:M**

Если между объектами предметной области имеется связь 1:M, то можно, как и в случае связи M:M, использовать отдельную связующую таблицу (рис. 3.4б – вариант 2). В отличие от связи M:M ключом связующей таблицы будет только идентификатор объекта, к которому направлен «единичный» конец связи.



а) Фрагмент ER-модели

Вариант 1
R1 (ИО1,С1)
R2 (ИО2, С2, ИО1)

Вариант 2
R1(ИО1,С1)
R2 (ИО2, С2)
RCB (ИО2, ИО1)

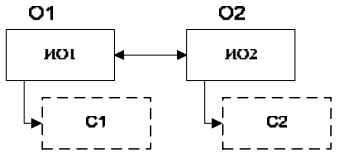
б) варианты отображения в реляционную модель

Рис. 3.4. **Отображение связи 1:M**

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), при этом надо дополнительно добавить идентификатор связанного с ней объекта (рис. 3.4б – 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. 3.4б – вариант 2).

Наличие между объектами связи типа 1:1 является довольно редкой ситуацией в реальной жизни. В принципе, если связь между объектами 1:1 и класс принадлежности обеих сущностей является обязательным, то для отображения обоих объектов и связи между ними можно использовать одну таблицу (рис. 3.5б – 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 из объектов, находящихся в связи 1:1, в дальнейшем потребуется отразить какие-то свои связи или в запросах часто требуется информация отдельно по каждому из объектов, тогда выбранное решение может усложнить или замедлить работу с БД.



а) Фрагмент ER-модели

Вариант 1
R1 (ИО1, C1, ИО2)
R2 (ИО2, C2)

Вариант 2
R1 (ИО1,C1)
R2 (ИО2, C2, ИО1)

Вариант 3
RОБ (ИО1, ИО2, C1, C2)

Вариант 4
R1 (ИО1,C1)
R2 (ИО2, C2)
RСВ (ИО1,ИО2)

б) Варианты отображения в реляционную модель

Рис. 3.5. **Отображение связи 1:1**

Если для каждого из этих объектов 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. 3.5б – варианты 1, 2).

Если класс членства одного из объектов является необязательными, 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 каждой из них является необязательным, то, чтобы избежать наличия пустых полей, следует исполь-

зовать три отношения: по одному для каждой сущности и одно – для отображения связи между ними (рис. 3.5б – вариант 4). В приведенном решении в качестве ключа связующей таблицы обозначен ИО1. С таким же успехом мог быть выбран ИО2.

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 даталогическую модель. Может быть в связи с этим она отсутствует в большинстве CASE-средств.

Естественным кажется путь, при котором в таблице базы данных, соответствующей объекту, к которому идут альтернативные связи, всем этим связям будет соответствовать одно поле, в котором 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 такое 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.

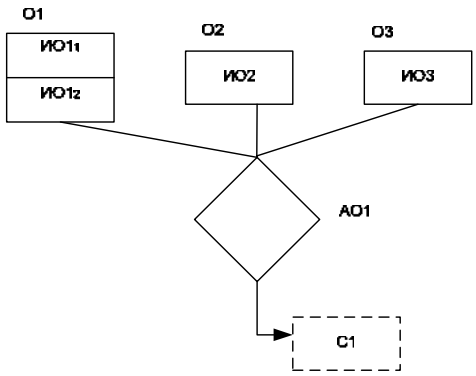
Часто объекты, объединенные альтернативной связью, по сути, являются подклассами обобщенного класса. Иногда имеет смысл по-другому изобразить ER-модель, чтобы увидеть другие варианты проектных решений.

Выше мы рассматривали 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.



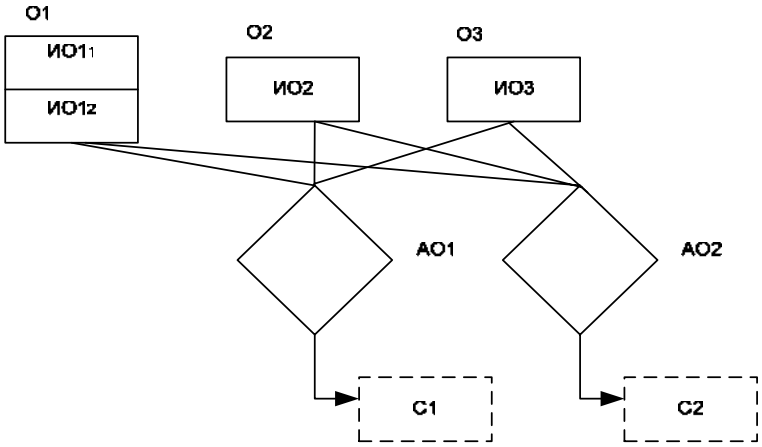
а) Фрагмент ER-модели

RAO1 (ИО1, ИО2, ИО3, С1)

б) Отображение в реляционную модель

Рис. 3.6. **Отображение агрегированного объекта**

Объединить информацию о нескольких агрегированных объектах в одно реляционное отношение можно только в том случае, если те объекты, с которыми связан каждый из них, полностью совпадают. Это является необходимым, но не достаточным условием для такого объединения. В каждом конкретном случае возможность и необходимость такого объединения надо определять особо.



а) Фрагмент ER-модели

Вариант 1
RAO1 (ИО1, ИО2, ИО3, С1)
RAO2 (ИО1, ИО2, ИО3, С2)

Вариант 2
RAO1 (ИО1, ИО2, ИО3, С1,С2)

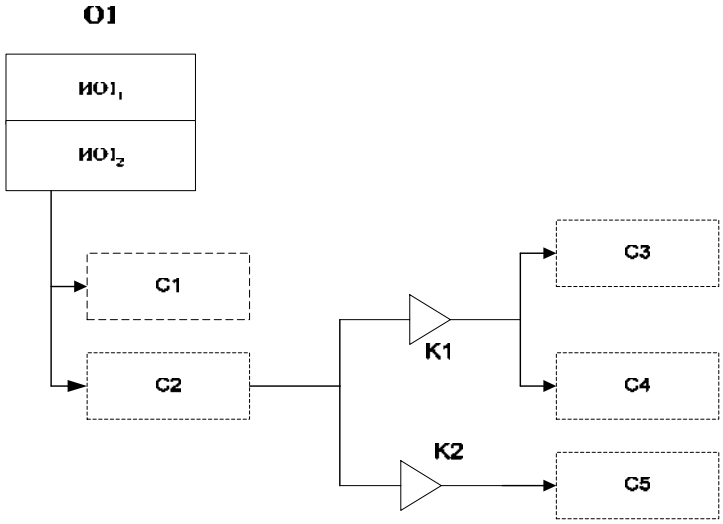
б) Отображение в реляционную модель

Рис. 3.7. **Отображение нескольких агрегированных объектов, имеющих одинаковые связи**

Отображение обобщенных объектов

При отображении обобщенных объектов могут быть приняты разные решения.

Во-первых, всему обобщенному объекту может быть поставлена в соответствие одна таблица базы данных (рис. 3.8б – вариант 1). В этом случае атрибутами этой таблицы будут идентификаторы обобщенного объекта, все единичные свойства, присущие объектам хотя бы одной категории, включая свойство, по которому производится разбиение на подклассы. Ключом таблицы будет один из идентификаторов этого объекта.



а) Фрагмент ER-модели

Вариант 1

R (ИО$_1$, ИО$_2$, С1, С2, С3, С4, С5)

Вариант 2

R1 (ИО$_1$, ИО$_2$, С1, С3, С4)
R2 (ИО$_1$, ИО$_2$, С1, С5)

Вариант 3

R1 (ИО$_1$, ИО$_2$, С1,С2)
R2 (ИО$_1$,С3, С4)
R3 (ИО$_1$, С5)

б) Варианты отображения в реляционную модель

Рис. 3.8. **Изображение обобщенного объекта**

Другим «крайним» вариантом является решение, при котором каждой из категорий объектов нижнего уровня ставится в соответствие отдельное отношение (рис.3.8б – 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 производится 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. 3.8б – вариант 3).

Другим вариантом проектного решения для отображения обобщенного объекта является использование так называемого «кодированного формата файла», при котором, как и варианте 1, используется одна таблица, но для всех «видовых» свойств каждого из подклассов выделяется одно поле, содержимое которого распознается по значению свойства, по которому производится разбиение класса на подклассы.

Перечень вариантов можно продолжить. Выбор конкретного решения будет зависеть от многих факторов, в том числе, насколько часто информация о разных категориях объектов обрабатывается совместно, как велико различие в «видовых» свойствах и некоторых других факторов.

Приведенный выше алгоритм основывается на предположении, 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. Для пересекающихся классов нельзя без модификации использовать вариант а) (так как признак классификации у каждого из экземпляров объекта может иметь несколько значений), но может быть использован вариант б).

Кроме того, при выборе проектного решения необходимо учитывать, является ли 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.

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.
- Информация о степени пересечения классов объектов (граф пересечений) может учитываться при выборе варианта отображения обобщенного объекта, а также при контроле целостности базы данных.

3.4.3. Дополнительные рекомендации по проектированию БД

Реляционная база данных, полученная в результате использования предложенного выше алгоритма проектирования, будет являться нормализованной и автоматически находится в четвертой нормальной форме.

В принципе, вся предметная область может быть представлена в виде одного универсального отношения. Недостатки такого способа хранения известны, и нормализация отношений служит их устранению. 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 в некоторых случаях 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 оказывают влияние много факторов, и не только напрямую связанные с оценкой структуры данных (объем занимаемой памяти, степень дублирования данных), но и с затратами времени на выполнение операций разного типа, а также 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 достаточно 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, проекции над горизонтальными сечениями эквивалентны выполнению этих операций над всем отношением и могут выполняться 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. Основным потребителем 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 вам, например, потребуется получить средний балл успеваемости всех студентов в последнюю сессию, то вам либо придется объединить соответствующие файлы, либо использовать достаточно сложный алгоритм вычисления требуемого показателя.

Горизонтальное разбиение отношения может производиться 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 др.

На это стоит 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 структу-

рированных баз данных. При этом в алгоритме перехода от ER-модели к модели, поддерживаемой конкретной СУБД, должны быть учтены особенности модели данных целевой СУБД.

3. Так как 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 создаваемых проектов, а при создании крупных корпоративных систем является практически неизбежным.

5. Использование 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 отображается 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 том, какие показатели следует хранить в базе данных?
17. В каком случае надо производить вертикальное и горизонтальное разбиение файлов/таблиц базы данных?

Глава 4.

Проектирование баз данных с использованием AllFusion ERwin Data Modeler

4.1. Общие сведения

Базы данных, как и другие информационные системы (ИС), проходят разные этапы своего жизненного цикла, начиная от замысла системы, предпроектного обследования, включая этапы проектирования, эксплуатации, а далее – модернизации системы.

Проектирование автоматизированной информационной системы (АИС) может начинаться «с нуля», когда для данной предметной области автоматизированной системы ранее не существовало или ее наличие игнорируется при создании новой системы. Вновь создаваемая система может встраиваться в уже существующую АИС. Может осуществляться процесс переноса существующей АИС в другую среду (для баз данных – использование другой СУБД). Каждая из указанных ситуаций накладывает отпечаток на процесс проектирования и, как следствие, на возможные режимы использования инструментальных средств проектирования.

Создание крупных проектов практически невозможно без использования средств автоматизации проектирования (CASE-систем). Их использование позволяет не только ускорить работы и повысить качество их выполнения, но и дает инструменты для организации коллективного труда группы проектировщиков.

Использование инструментальных средств при проектировании баз данных затрагивает разные этапы жизненного цикла АИС. Оно в определенной мере предопределяет процесс обследования и дает инструмент для отображения его результатов.

Наибольшее распространение в настоящее время получили системы, которые позволяют с помощью графических языков отобразить предметную область (построить концептуальную модель) и затем осуществить автоматический переход от концептуальной модели к модели данных в среде выбранной целевой СУБД.

Использование CASE-систем такого типа объединяет не только проектировщиков АИС, но и заказчиков системы, и поэтому отдельные механизмы, а именно нотации, используемые на этапе концептуального моделирования системы, должны грамотно восприниматься всеми ими.

Различают прямое проектирование (forward-engineering) – процесс получения структуры базы данных для выбранной целевой СУБД на основе построенной ER-модели, и обратное проектирование (reverse-engineering – реверс-инжиниринг) – когда ER-модель получается на основе существующей базы данных. CASE-средства обычно поддерживают оба эти процесса.

М. Фаулер в книге «UML.Основы», говоря о способах применения универсального языка моделирования UML, отмечает, что различные схемы, построенные с использованием языков графического моделирования, могут использоваться в разных режимах: «режим эскиза, режим проектирования и режим языка программирования» (СПб.: Символ-Плюс, 2004). Это же относится и к использованию систем моделирования данных. Более того, CASE-средства обычно предоставляют возможности представления моделей разных уровней, что позволяет выбрать подходящий механизм моделирования для каждого из режимов/стадий проектирования.

Проектирование баз данных с использованием автоматизированных инструментальных средств является одним из разделов курса «Базы данных». При изучении курса могут использоваться любые программные продукты данного класса, представленные на рынке, например, PowerDesigner, ER/Studio, AllFusion ERwin Data Modeler (ранее эта компонента AllFusion выпускалась в виде отдельного продукта и называлась ERwin) и другие. Многие из этих систем имеют сходную функциональность и даже базируются на одних и тех же стандартах изображения ER-моделей, а именно – методологии IDEF1X.

В данном учебном пособии рассматриваются вопросы проектирование баз данных с использованием инструментального средства автоматизации проектирования **AllFusion** ERwin Data Modeler 4.1.4. Интерфейс и функциональные возможности этой системы значительно отличаются от ERWin 3.5.2. Существенных же отличий ERWin 4.0. от AllFusion ERwin Data Modeler 4.1.4 не наблюдается.

AllFusion ERwin Data Modeler 4.1.4 входит в комплексное инструментальное средство проектирования AllFusion Modeling Suite фирмы CA.

В дальнейшем по тексту данного учебного пособия для краткости будем использовать название ERWin, имея в виду AllFusion ERwin Data Modeler 4.1.4.

ERWin является инструментальным средством, позволяющим автоматизировать процесс проектирования реляционных баз данных. Для нормального восприятия материала данного учебного пособия необходимо хорошо понимать сущность реляционной модели данных, уметь определять первичные ключи отношений и альтернативные ключи, понимать, когда следует, а когда не надо задавать альтернативные ключи при описании модели.

4.2. Выбор шаблона представления модели

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

Для создания новой модели после запуска системы надо выбрать позицию ***Create a new model*** (рис. 4.1).

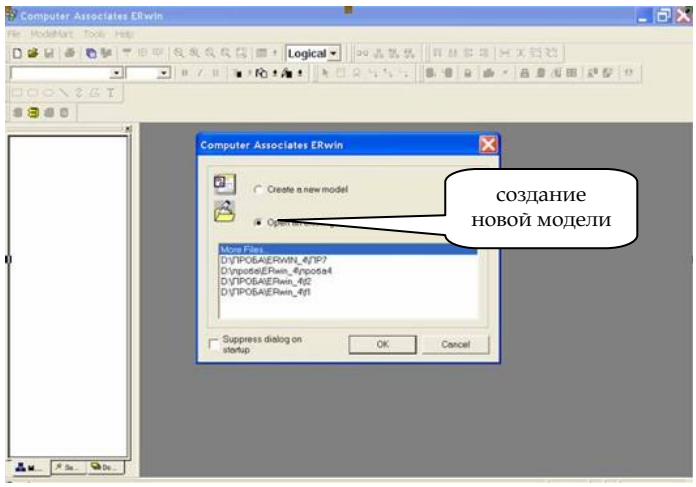


Рис. 4.1. **Начальный экран**

При моделировании в среде ERWin можно выбрать одну из трех возможностей: создавать только логическую модель, только физическую или и логическую и физическую модель одновременно. На рис. 4.2 представлен вид экрана при выборе типа модели «Logical».

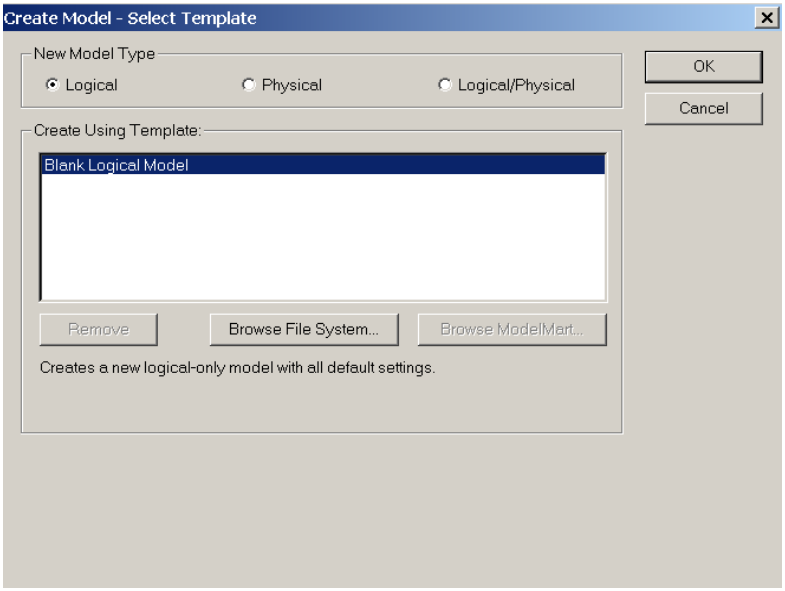


Рис. 4.2. **Выбор типа модели. Экран при выборе типа модели «Logical»**

При выборе «Physical» или «Logical/Physical» экран будет иметь вид, представленный на рис. 4.3. В этом случае уже сразу перед началом моделирования можно выбрать целевую СУБД (Target Database). Целевую СУБД можно выбирать/изменять и на более поздних этапах проектирования.

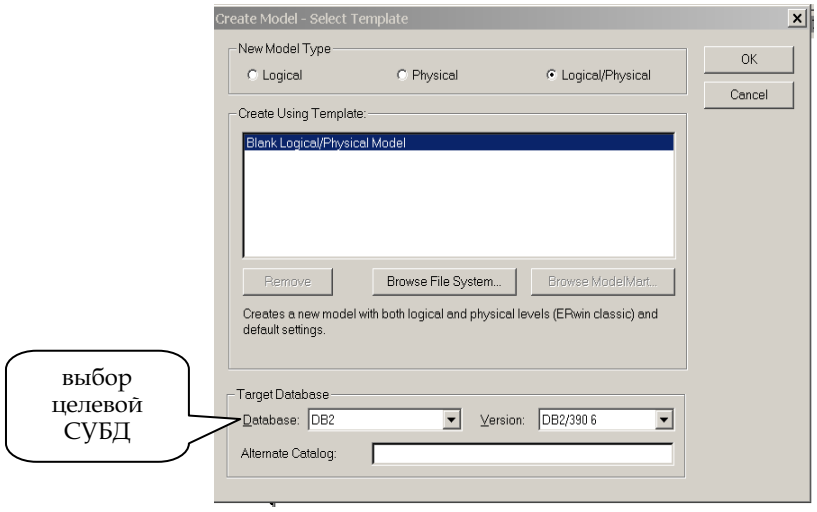


Рис. 4.3. **Экран при выборе типа модели «Physical» или «Logical/Physical»**

Как правило, создание модели данных начинается с создания ER-модели логического уровня, если речь не идет о реверс-инжиниринге, когда логический уровень модели получается на основе существующей базы данных.

Создание отдельно логической и физической моделей может быть полезно при проектировании распределенных гетерогенных систем, когда одной логической модели можно поставить в соответствие несколько физических.

Мы выберем вариант «Logical/Physical». Экран после этого выбора будет иметь вид, представленный на рис. 4.4. При работе в таком режиме можно в любой момент переключаться между логической и физической моделью.

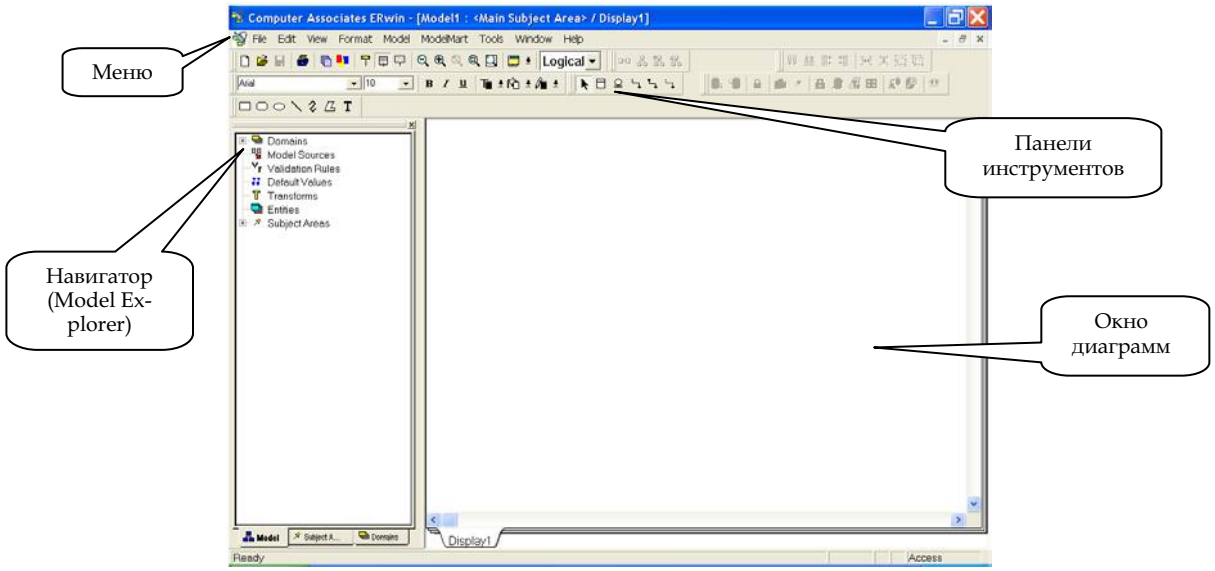


Рис. 4.4. **Основной экран при разработке модели ERWin**

Если был выбран режим «совмещенной» «Logical/Physical» модели, позволяющий переход от логической модели к физической и обратно, то в любой момент можно получить отдельно логическую и физическую модели, воспользовавшись возможностью разделения (Split) исходной модели (меню ***Tools/Split L/P Model***). При этом система предложит последовательно задать имя логической и физической модели.

ERWin автоматически сохраняет историю всех изменений, связанных с моделью в целом или тем или иным ее объектом. В описании истории создания модели, полученной описанным выше путем, автоматически будет сформировано сообщение «Created by splitting model [имя модели] – «создано путем расщепления модели [имя модели] (рис. 4.5).

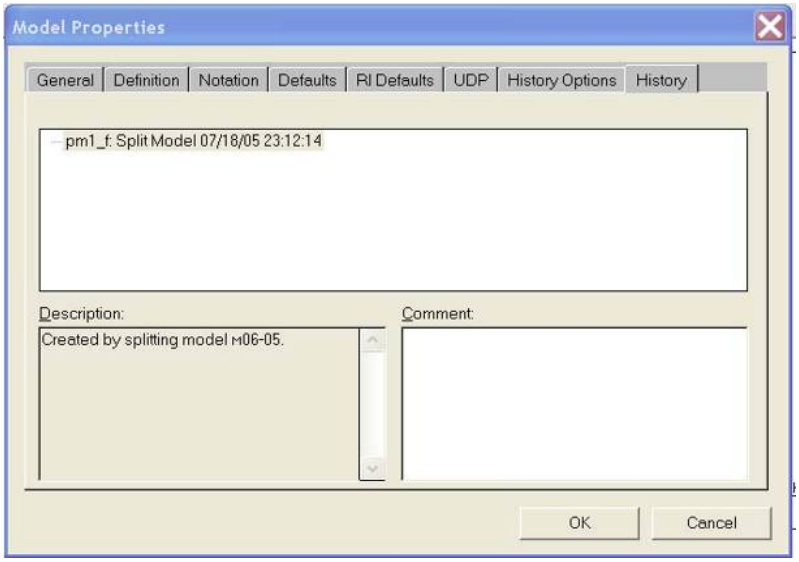


Рис. 4.5. **Окно Model Properties (Свойства модели). Вкладка History (История).**

4.3. Интерфейс ERWin

В рабочей области ERWin (см. рис. 4.4) можно выделить несколько зон: меню, панели инструментов, навигатор (Model Explorer), окно диаграмм (область моделирования).

Интерфейс ERWin включает восемь перемещаемых панелей инструментов, список которых доступен из меню ***View/Toolbars*** (рис. 4.6). Их назначение будет рассматриваться по мере рассмотрения соответствующих вопросов.

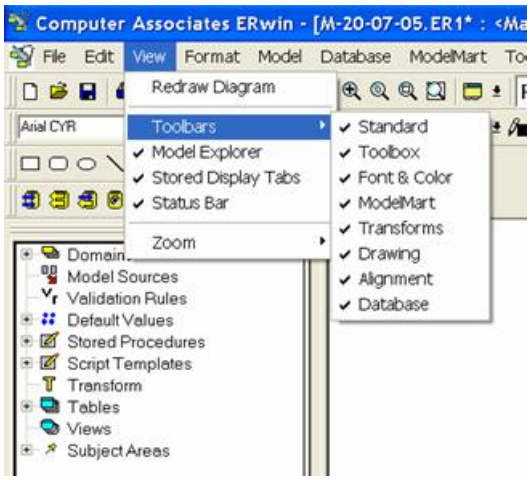
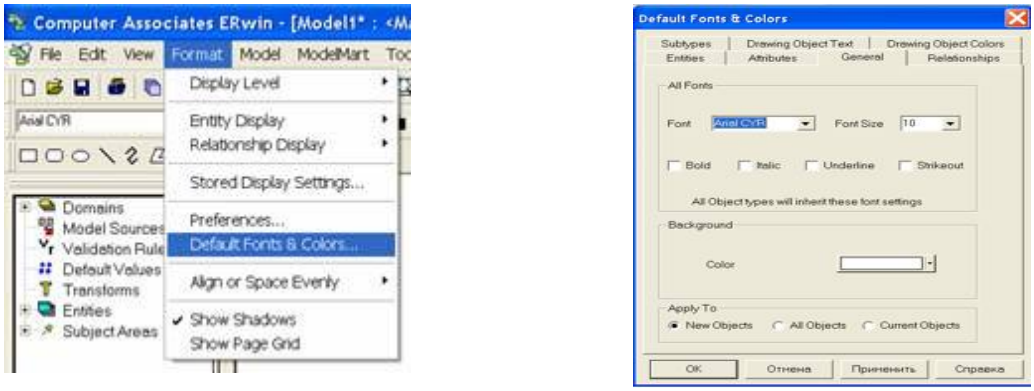


Рис. 4.6. **Список панелей инструментов**

Панель Model Explorer содержит в виде раскрывающихся списков все объекты модели. Перечень объектов будет несколько различаться в зависимости от того, с какой моделью (логической или физической) идет работа. Позиционировавшись на нужном объекте в списке, можно щелкнуть правой кнопкой мыши и в появившемся контекстном меню выбрать требуемое действие.

4.4. Выбор шрифтов

При изображении ER-модели можно использовать различные шрифты. Так как логическая ER-модель используется на разных этапах создания базами данных разными категориями пользователей, то желательно, чтобы она была представлена на естественном (в нашем случае – русском) языке. Чтобы определить используемые шрифты, можно выбрать позицию в меню ***Format/Default Font & Colors*** (рис. 4.7а).



а) Шаг 1 **б) Шаг 2**

Рис. 4.7. **Выбор шрифтов, используемых по умолчанию**

В появившемся окне (рис. 4.7б) следует выбрать какой-либо из кирилистических шрифтов.

Аналогичного результата можно достигнуть и воспользовавшись панелью Font&Color (рис. 4.8).

Рис. 4.8. **Панель Font&Color**

4.5. Нотации, используемые при построении ER-моделей

При построении ER-модели в ERWin можно выбрать нотацию, которая будет использоваться при изображении данной модели. Для того чтобы осуществить переключение между нотациями, следует выбрать позиции меню ***Model/Model Properties*** и в появившемся окне **Model Properties** выбрать закладку ***Notation***. Как видно из рис. 4.9, для логического уровня представления ER-модели можно осуществлять выбор из двух нотаций: IDEF1X (Integration DEFinition for Information Modeling) и IE (Information Engineering).

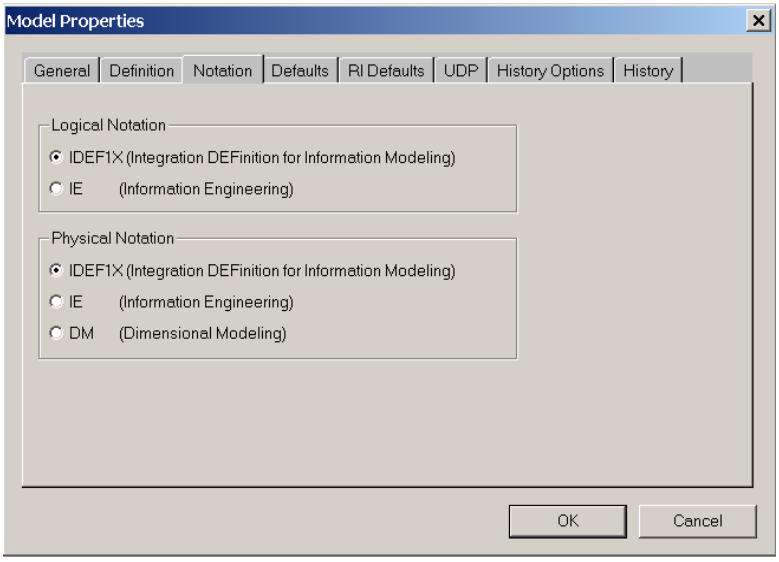


Рис. 4.9. **Выбор нотации**

Вид палитры инструментов (ERWin Toolbox) для случая использования нотации IDEF1X представлен на рис. 4.10.

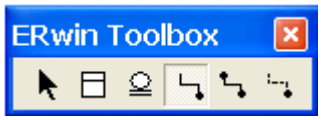


Рис. 4.10. **Палитра инструментов при использовании нотации IDEF1X**

Если палитра инструментов на экране отсутствует, следует в меню ***View/Toolbars/Toolbox*** соответствующую строку отметить знаком √.

Рассмотрим назначение кнопок палитры инструментов. Кнопка  используется при создании новой сущности.

Кнопки  соответствуют идентифицирующей связи, связи «многие-ко-многим» и неидентифицирующей связи (в порядке их расположения на панели). Множественный конец связи обозначен точкой.

Кнопка  используется при создании Категорий (это соответствует понятию «обобщенного объекта» в базовой ER-модели[1]. Этот знак используется, когда в модели надо отобразить родо-видовые связи между сущностями.

Вид палитры инструментовдля случая использования нотации IE представлен на рис. 4.11.

Рис. 4.11. **Палитра инструментов при использовании нотации IE**

[1] Используется, в частности, в книге С.М. Диго «База данных: проектирование и использование».

Большинство различий в этих нотациях относятся к несущественным различиям между представлениями моделей. Если сравнивать обозначения на палитрах инструментов нотаций IDEF1X и IE, видно, что разница заключается в том, что для обозначения множественного конца линии связи в нотации IDEF1X используется точка, а в нотации IE – «лапка».

Разница при описании обобщенных объектов является более существенной, так как заключается не только в том, что для обозначения Категории используется другой графический символ (), но и вводятся разные понятия («полная» и «неполная» категория в нотации IDEF1X, «эксклюзивная» и «неэксклюзивная» категория в нотации IE). Более подробно различия методик моделирования IDEF1X и IE будут рассмотрены далее.

Модель, представленная в одной нотации, может быть легко автоматически преобразована в другую нотацию.

Далее мы рассмотрим создание логической модели в нотации IDEF1X. Эта нотация выбирается по умолчанию.

4.6. Построение логической модели

4.6.1. Сущности

В работе «Базы данных: проектирование и использование» были выделены несколько разновидностей объектов (сущностей). 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. Выделяют несколько разновидностей сложных объектов: составные, обобщенные и агрегированные объекты.

Составной объект соответствует отображению отношения «целое – часть».

Обобщенный объект отражает наличие связи «род – вид» между объектами предметной области.

Агрегированные объекты соответствуют обычно какому-либо процессу, в который оказываются «вовлеченными» другие объекты.

В ERWin имеются специальные условные обозначения для изображения простых сущностей и специальный символ, используемый при изображении обобщенных объектов (хотя само понятие «обобщенный объект» в ERWin отсутствует). Остальные разновидности сущностей можно отобразить, используя соответствующие сочетания обозначений простых сущностей и связей между ними.

4.6.2. Создание простых сущностей

Для создания новой сущности следует воспользоваться кнопкой  .

После позиционирования на экране и нажатия на правую кнопку мыши появится значок, отображающий сущность (рис. 4.12). В выделенную вверху область следует ввести имя сущности.

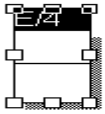


Рис. 4.12. **Создание сущности. Шаг 1**

После этого на экране появится область для ввода имени атрибута этой сущности (рис. 4.13). Можно последовательно ввести таким образом имена нескольких атрибутов описываемой сущности. Эти атрибуты будут помечены как входящие в первичный ключ. Исходя из этого можно порекомендовать: прежде чем описывать сущность, продумать, какой (какие) атрибут(ы) следует выбрать в качестве первичного ключа, и начать создание сущности с описания именно этих атрибутов. После ввода атрибутов, входящих в состав первичного ключа, ввод атрибутов следует временно прекратить. На экране появится изображение создаваемого объекта. Его следует открыть двойным щелчком левой клавиши мыши. После чего на экране появится окно **Attributes** (рис. 4.15), в котором и надо продолжить описание остальных атрибутов.

Рис. 4.13. **Возможность ввода имен атрибутов при создании сущности**

Но можно поступить и по-другому, а именно: после задания имени сущности завершить ввод, выделить появившийся на экране значок, соответствующий этой сущности, и либо дважды щелкнуть по нему левой клавиши мыши, либо нажать на правую клавишу мыши и в появившемся контекстом меню (рис. 4.14) выбрать позицию «Atributes».

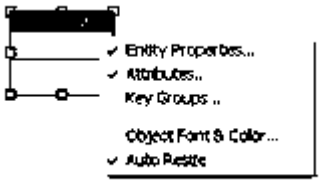


Рис. 4.14. **Контекстное меню. Переход в редактор атрибутов**

Далее в появившемся окне **Attributes** (рис. 4.15) следует ввести все необходимые атрибуты и задать для каждого из них требуемые характеристики.

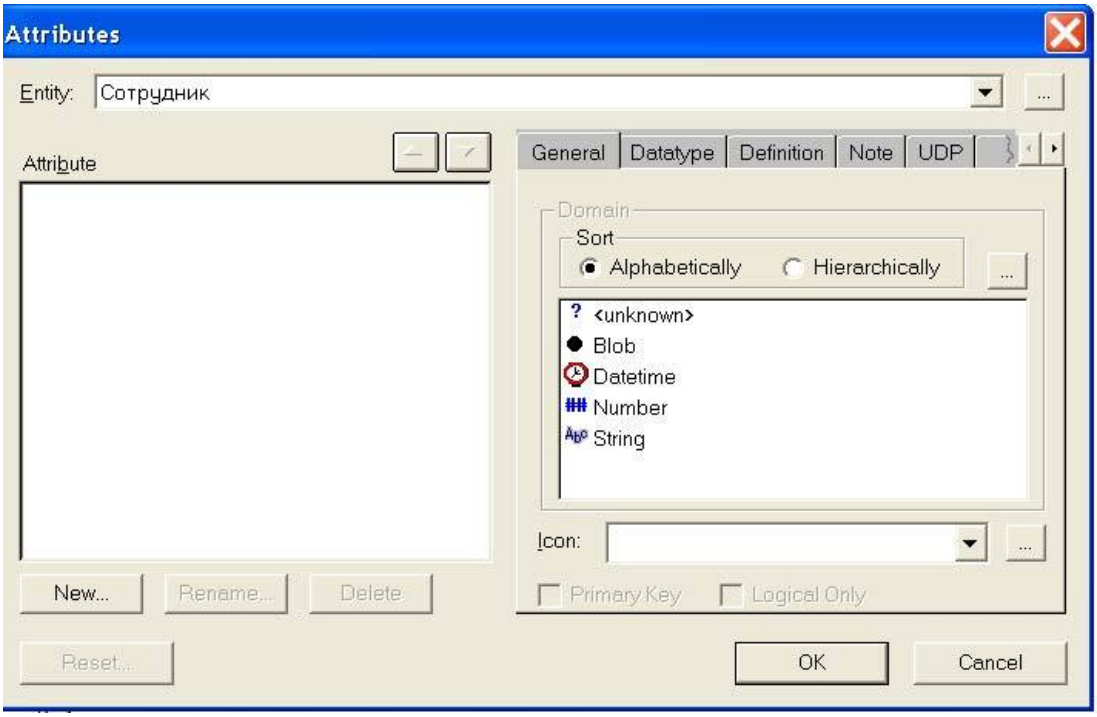


Рис. 4.15. **Окно Attributes**

Для описания нового атрибута следует нажать кнопку ***New.*** Она является единственно активной, если ни один атрибут еще не описан. В появившемся окне **New Attribute** (рис. 4.16) следует задать имя атрибута (Attribute Name) и домен. Соотнесение атрибута с доменом будет использоваться при определении типа данных при переходе от логической модели к физической.

Задание имени колонки (Column Name) не является обязательным и используется тогда, когда проектировщик хочет задать имя поля в таблице базы данных, отличающееся от имени атрибута. 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 полей (длина, допустимость пробелов и т.п.).

Признак ***Logical Only*** следует использовать тогда, когда атрибут, присутствующий в логической модели, не будет переноситься в физическую модель. Такая ситуация, может, например, возникать тогда, когда в логической модели отображаются вычисляемые показатели, которые в физической модели отражаться не будут.

В окне **New Attribute** можно также задать домен для атрибута. Домен определяет допустимые типы данных.

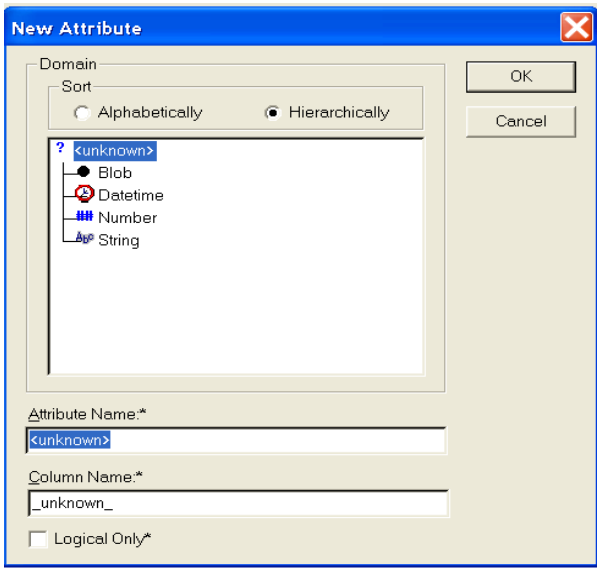


Рис. 4.16. **Окно New attributes**

После завершения описания атрибута в окне **New Attribute** система возвращается в окно **Attributes**. Находясь в этом окне, можно отредактировать ранее введенную информацию.

Для ключевых полей в окне **Attributes** можно задать признак ***Primary Key*** (или отменить его, если вы изменили решение относительно выбора первичного ключа).

На рис. 4.17 представлено описание атрибута «Табельный номер». В качестве имени атрибута было выбрано «Таб_ном», а домен – Number. Имя колонки – не задавалось.

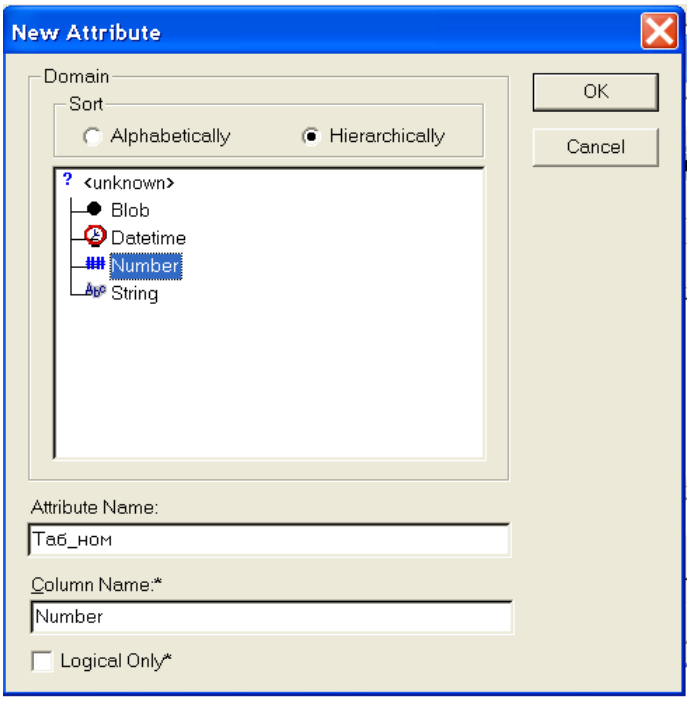


Рис. 4.17. **Описание атрибута**

Атрибут «Таб_ном» является первичным ключом сущности «Сотрудник». Поэтому в окне *Attributes* для этого атрибута надо задать признак ***Primary Key*** (рис. 4.18).

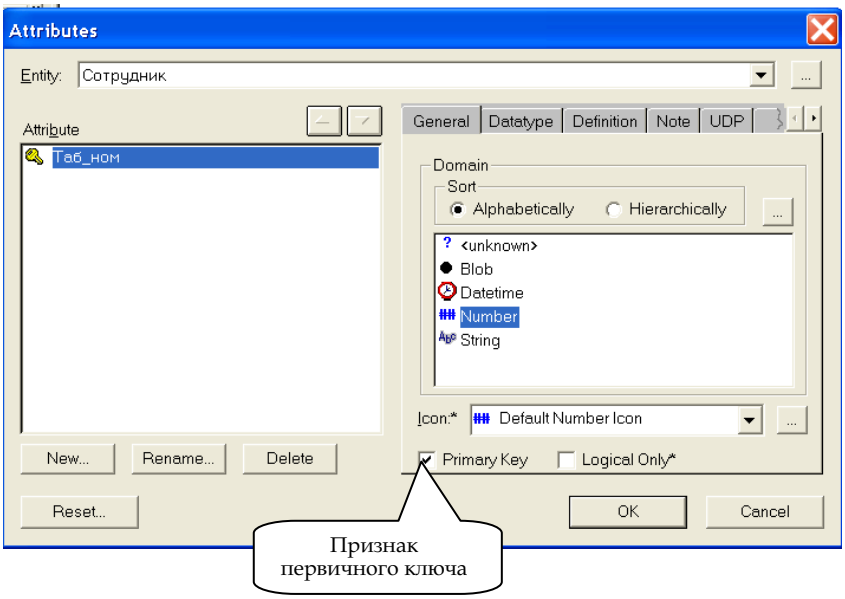


Рис. 4.18. **Определение первичного ключа**

Приведенный ниже пример (рис. 4.19) иллюстрирует применение признака **Logical Only**: признак «только логический» указан для атрибута «Возраст», который не следует хранить в базе данных.

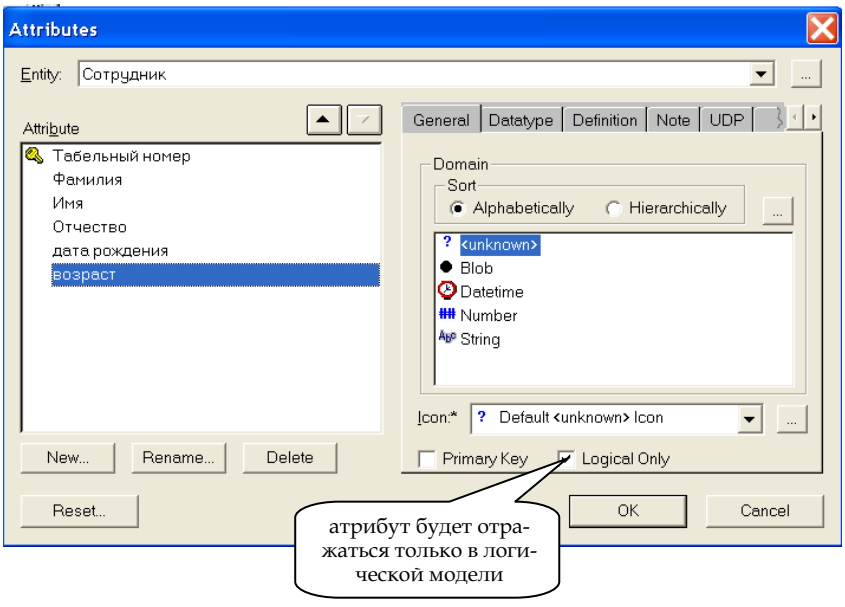


Рис. 4.19. **Использование признака Logical Only**

4.6.3. Дополнительные свойства атрибутов. Создание ключей и инверстных входов

При создании реляционных баз данных по отдельным полям или совокупностям полей могут быть построены индексы. По умолчанию по полям, выбранным в качестве первичного ключа, всегда 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 произвести индексирование. Если поле, по которому производится индексирование, является неуникальным, то оно называется ***инверсным входом*** (Inversion Entry).

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.

В ERWin для задания первичного или альтернативного ключа, а также инверсного входа можно, позиционировавшись на значке сущности, нажать правую кнопку мыши и в появившемся контекстном меню (см. рис. 4.14) выбрать позицию ***Key Group.*** Либо выбрать вкладку ***Key Group*** в окне **Attributes** (рис. 4.20).

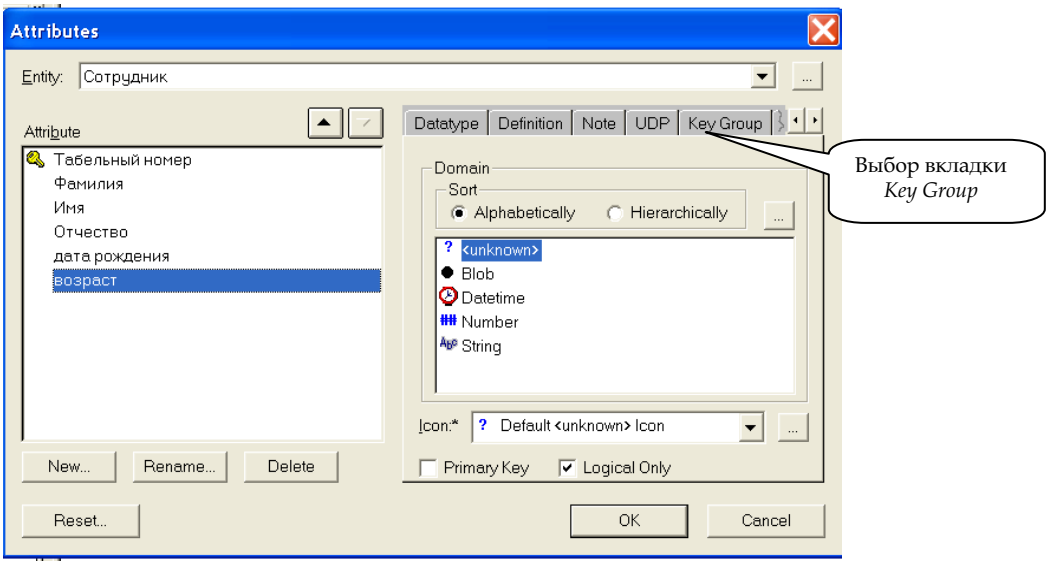


Рис. 4.20. **Окно Attributes. Выбор вкладки Key Group**

После этого в окне **Key Groups** (рис. 4.21) следует нажать кнопку ***New***.

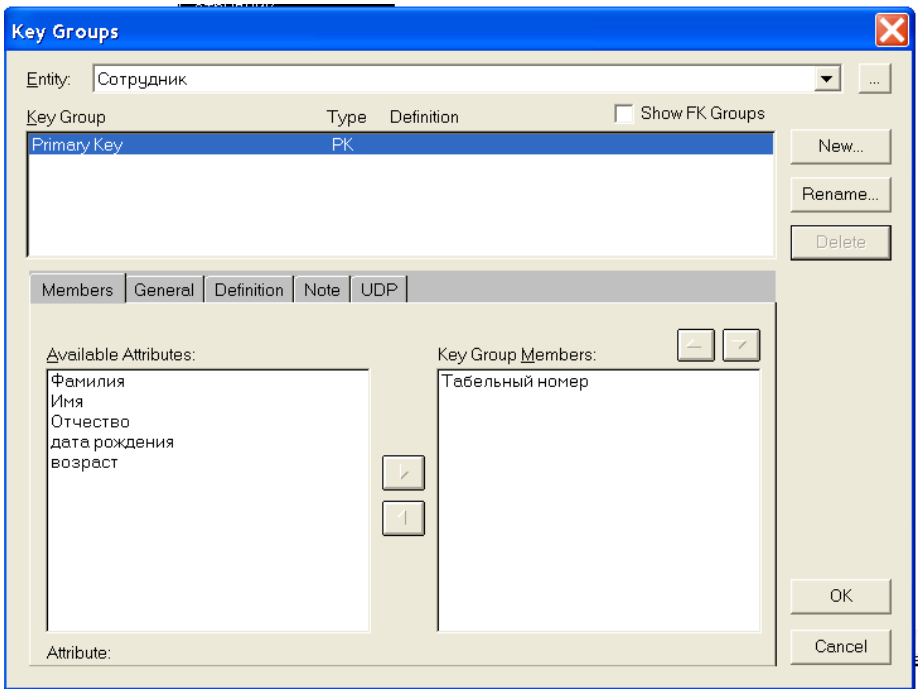


Рис. 4.21. **Создание новой ключевой группы (Key Group). Шаг 1**

В появившемся окне **New Key Group** (рис. 4.22) следует выбрать тип ключевой группы (***Key Group Type***). Предположим, что мы хотим создать составной инверсный вход, включающий атрибуты Фамилия, Имя, Отчество. Тип ключевой группы будет соответственно определен как ***Inversion Entry***.

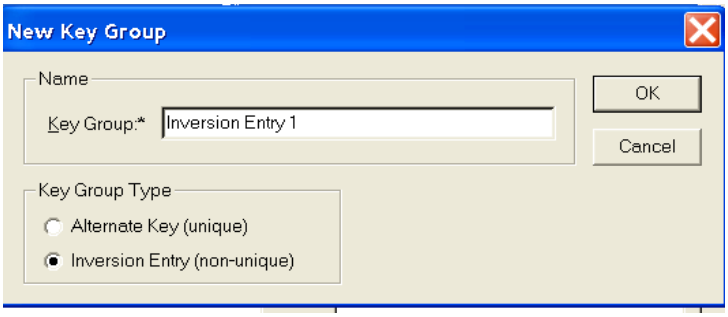


Рис. 4.22. **Создание инверсного входа**

После нажатия на кнопку OK появится окно **Key Groups** (рис. 4.23) с новой строкой, соответствующей вновь создаваемой группе. Далее в списке доступных атрибутов (Available Attributes) надо позиционироваться на первый из тех атрибутов, которые должны входить в состав создаваемой группы, и, воспользовавшись кнопкой  , переместить его в окно ***Key Group Members***. Так последовательно надо поступить и с другими атрибутами, входящими в данную группу.

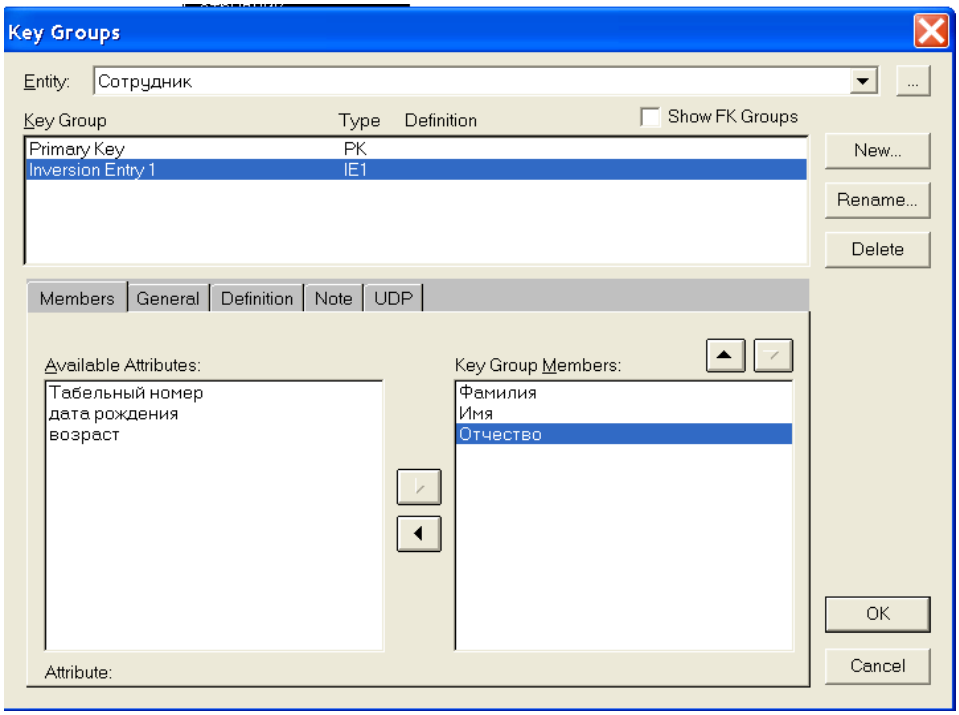


Рис. 4.23. **Создание инверсного входа. Определение состава группы**

Ключевой группе можно дать новое имя. Для этого следует воспользоваться кнопкой **Rename** (переименовать). В появившемся окне ***Rename Key Group*** (рис. 4.24) надо ввести желаемое имя группы.

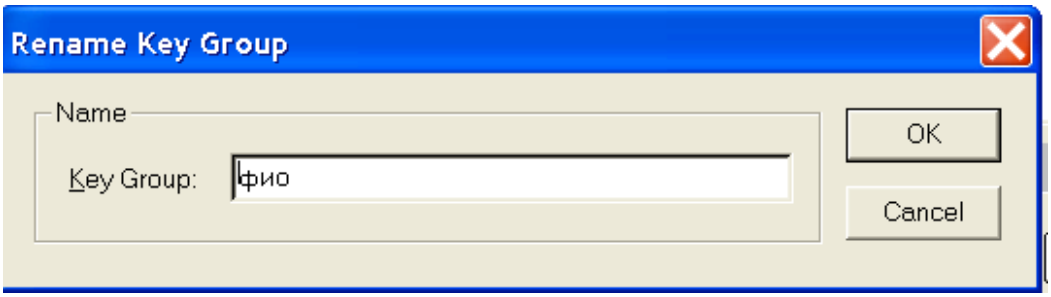


Рис. 4.24. **Задание имени группы**

После выполненных шагов окно ***Key Groups*** будет иметь вид, представленный на рис. 4.25.

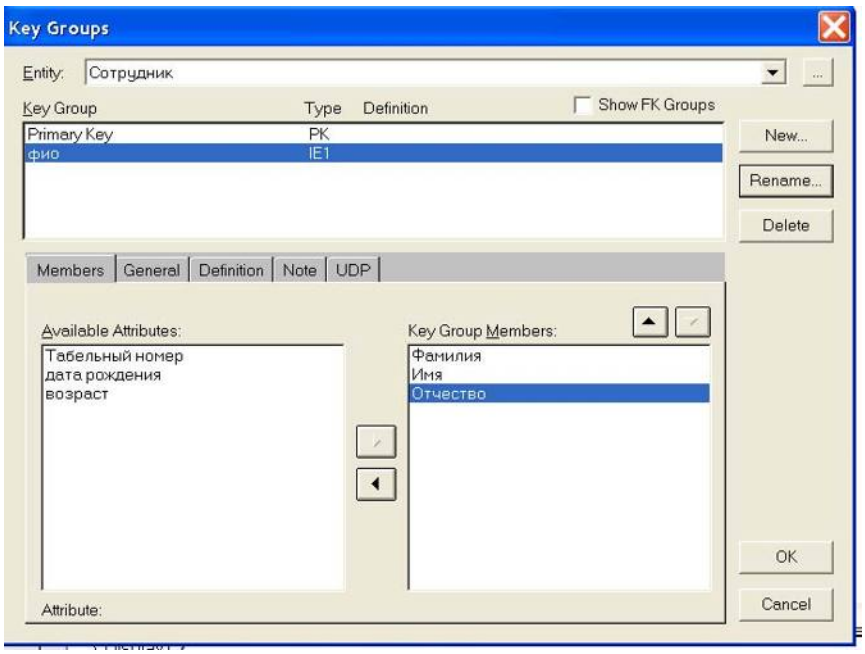


Рис. 4.25. **Создание инверсного входа. Вид окна после выполнения всех шагов**

Чтобы на схеме, отображающей ER-модель, были видны признаки первичного и альтернативного ключа, а также инверсного входа, надо в меню ***Format/Entity Display*** (рис. 4.26) выбрать соответствующие позиции списка.

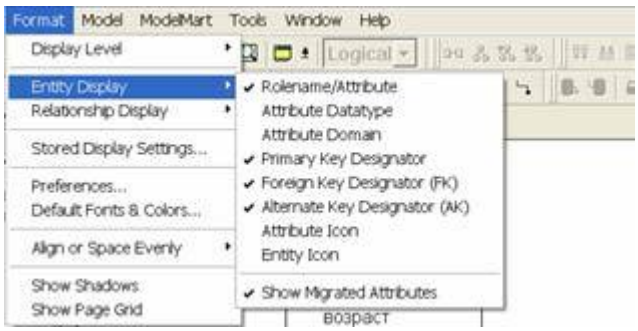


Рис. 4.26. **Задание параметров отображения сущности**

После этого сущность **Сотрудник** будет иметь вид, представленный на рис. 4.27.

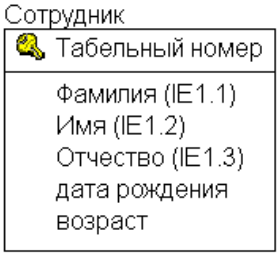


Рис. 4.27. **Сущность «Сотрудник».**
Задан первичный ключ и один составной инверсный вход

Для сущности может быть задано несколько инверсных входов. Причем один и тот же атрибут может входить в состав нескольких разных инверсных входов.

Например, мы можем создать еще один инверсный вход, включающий только атрибут Фамилия. После этого сущность будет иметь вид, представленный на рис. 4.28.

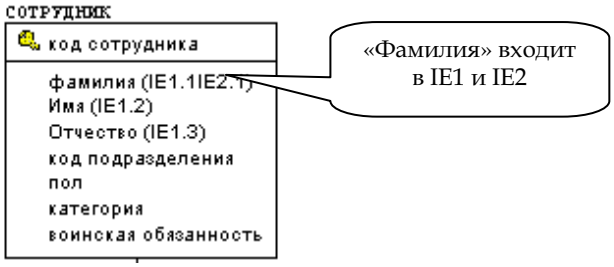


Рис. 4.28. **Сущность «Сотрудник».**
Задан первичный ключ и два инверсных входа

Приведенные выше примеры демонстрируют только возможности системы. Вопросы проектирования (в частности, когда и какие инверсные ключи следует создавать) – не рассматриваются.

Аналогично можно создать альтернативный ключ: в окне ***Key Groups*** (см. рис. 4.21) нажать кнопку **New**; в окне ***New Key Group*** (см. рис. 4.22) следует выбрать тип ключевой группы (Key Group Type) – Alternative Key. На рис. 4.29 в качестве примера был создан альтернативный ключ, включающий атрибуты **Фамилия-Имя-Отчество-Дата рождения.**

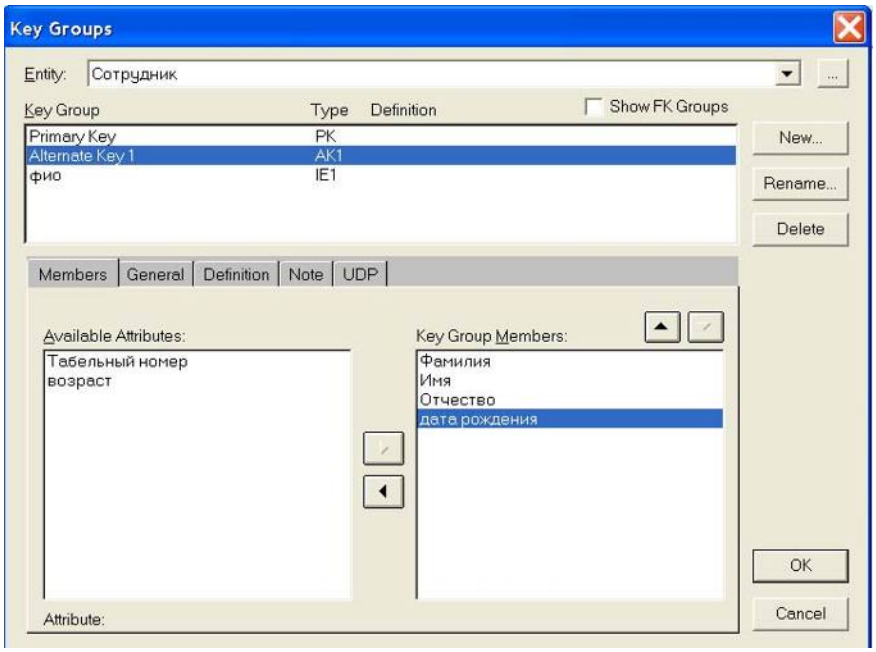


Рис. 4.29. **Создание составного альтернативного ключа**

После этого сущность **Сотрудник** примет вид, представленный на рис. 4.30.

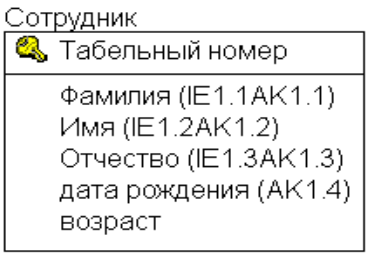


Рис. 4.30. **Сущность «Сотрудник».**
Задан первичный ключ, альтернативный ключ и инверсный вход

В ERWin ключи (первичные и альтернативные) и инверсные входы можно задавать как при описании логической модели, так и при создании физической модели. При переключении между физической и логической моделью вновь введенные характеристики будут отображаться в обоих представлениях безотносительно к тому, в каком режиме они были введены.

4.6.4. Дополнительные характеристики сущности

Описание сущности не ограничивается только заданием ее имени. Если в контекстном меню (см. рис. 4.14) выбрать позицию ***Entity Properties***, то в появившемся окне ***Entities*** (рис. 4.31) можно задать дополнительные характеристики сущности.

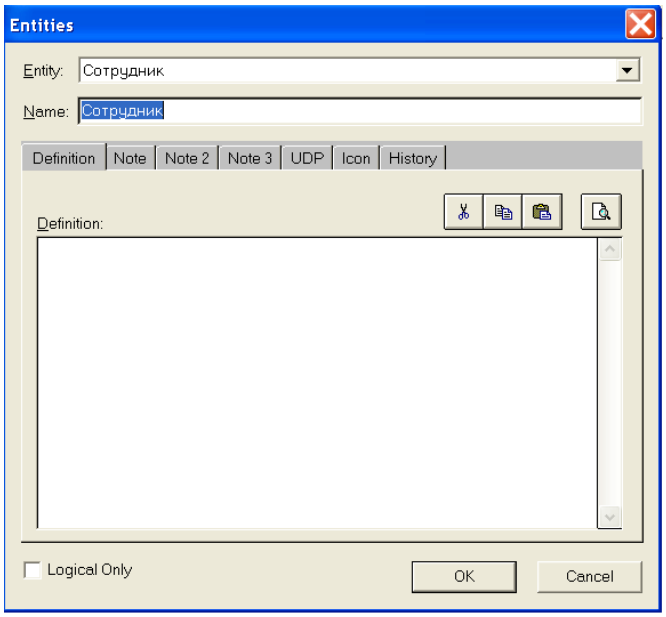


Рис. 4.31. **Окно Entities**

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. 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 поэтому подробно рассматриваться не будет.

4.6.5. Описание иерархии обобщения

В ERWin можно отображать родовидовые связи между сущностями. Прежде чем перейти к изложению того, как это надо делать, сделаем некоторые предварительные замечания.

В некоторых версиях ERWin при переходе от логической модели к физической ключ родительской сущности по умолчанию не мигрирует в подчиненную сущность. Для того чтобы это происходило, надо в меню ***Model*** выбрать позицию ***Model Properties*** и в появившемся окне отметить позицию ***Sypertype/Subtype with Identifying Relationships*** (рис. 4.32).

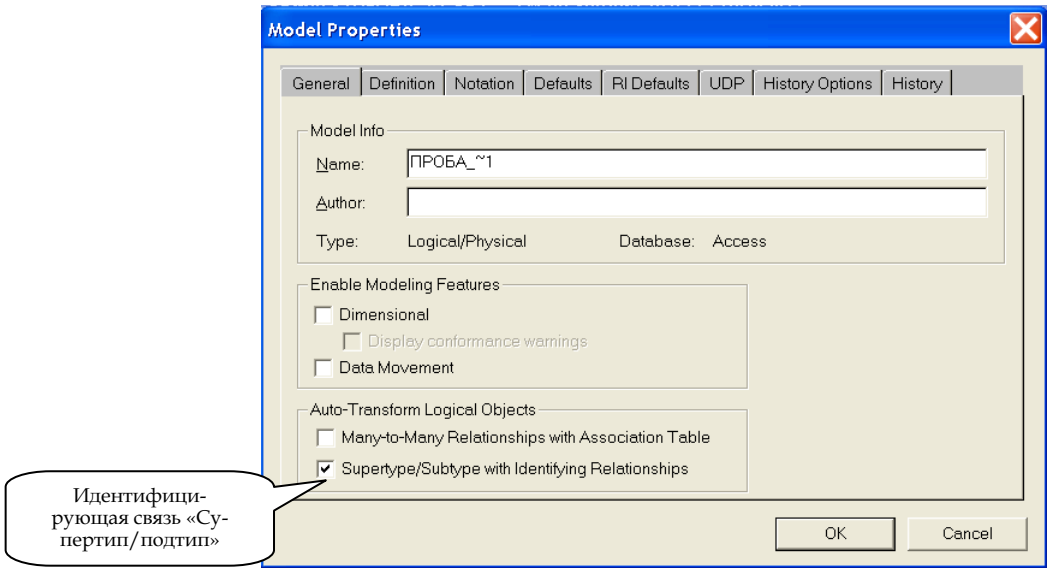


Рис. 4.32. **Выбор свойства модели Sypertype/Subtype with Identifying Relationships**

Как отмечалось ранее, возможности изображения иерархии обобщения в нотациях IDEF1X и IE несколько различаются. Рассмотрим сначала, как это выполняется в нотации IDEF1X.

4.6.5.1. Описание иерархии обобщения в нотации IDEF1X

Для изображения иерархии обобщения следует предварительно изобразить как родовой, так и видовые объекты.

Далее надо 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 выбрать на палитре инструментов знак идентифицирующей связи и затем щелкнуть по символу категории, а затем – по следующей видовой сущности.

На рис. 4.33. изображен фрагмент модели, на котором для сущности СОТРУДНИК выделены две категории: ШТАТНЫЕ и СОВМЕСТИТЕЛИ.

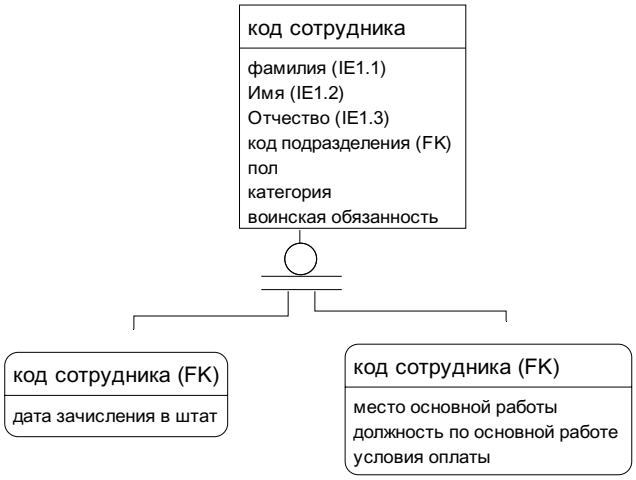


Рис. 4.33. **Пример иерархии обобщения в нотации IDEF1X**

Обобщенный объект в ERWin называют Иерархией Категорий. Различают полные (Complete sub-category) и неполные (Incomplete sub-category) категории. Полные категории означают, что любой экземпляр видовой сущности обязательно входит в какой-либо подкласс. Неполные категории означают, что могут быть сущности, не принадлежащие ни одному из изображенных в модели подклассов сущностей. На панели элементов имеется только знак полной категории (). Чтобы изменить знак категории необходимо позиционироваться на нем, щелкнуть правой кнопкой мыши и в контекстном меню выбрать позицию ***Subtype Relationship***. В появившемся окне (рис. 4.34) следует выбрать переключатель ***Incomplete***.

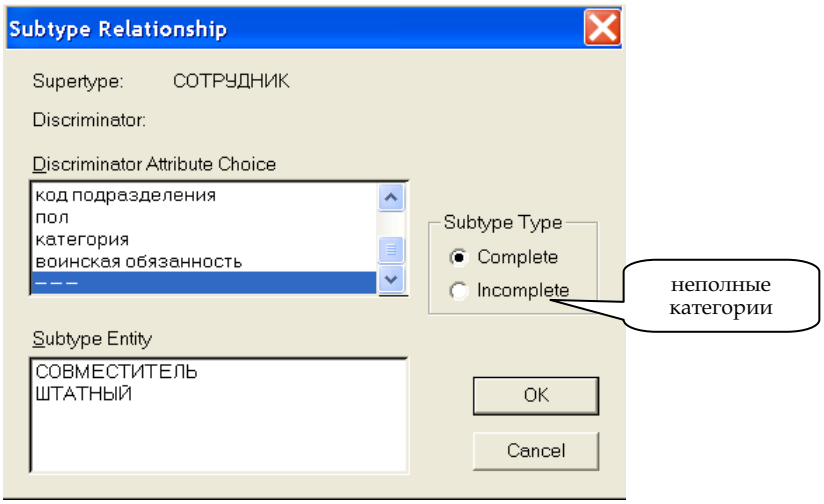


Рис. 4.34. **Создание неполных категорий**

После этих действий вместо знака полной категории на схеме появится знак неполной категории ().

Атрибут, по которому класс разбивается на подклассы, называется ***дискриминатором*** (Disriminator). ERWin позволяет указать в модели, какой из атрибутов является дискриминатором, и вывести его имя. Чтобы определить дискриминатор, соответствующий

атрибут должен присутствовать в описании родовой сущности. Далее в окне ***Subtype Relationship*** надо позиционироваться на этом атрибуте. Предположим, что в нашем примере разбиение идет по значения атрибута «категория» (рис. 4.35).

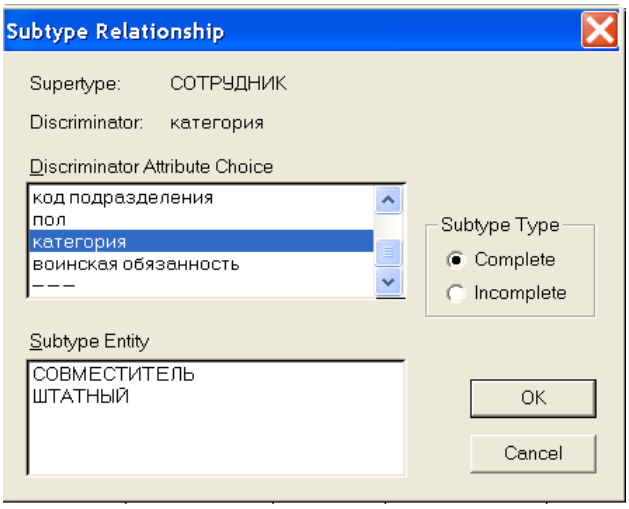


Рис. 4.35. **Определение дискриминатора**

После выполненных манипуляций фрагмент логической модели, отражающий обобщенный объект СОТРУДНИК, приобрел вид, изображенный на рис. 4.36.

Рис. 4.36. **Изображение иерархии категорий с заданным именем дискриминатора**

4.6.5.2. Описание иерархии обобщения в нотации IE

Как было указано выше, в нотации IDEF1X иерархия категорий может быть двух типов – полная и неполная. В нотации же IE происходит деление иерархий категорий на эксклюзивную и неэксклюзивную. Для обозначения Категорий в нотации IE используются графические символы, представленные на рис. 4.37.

а) эксклюзивная б) неэксклюзивная

Рис. 4.37. **Типы иерархии категорий в нотации IE**

Разница между нотациями заключается не только в используемых графических символах, но и в том, какие ситуации в предметной области позволяет отобразить каждая из этих нотаций. В нотации IE все категории считаются полными. Символ эксклюзивной категории следует использовать тогда, когда каждый экземпляр любой видовой сущности может входить только в один из подклассов. Неэксклюзивные категории означают, что экземпляр сущности может относиться к нескольким подклассам одновременно. Например, если при отображении сущности «Личность» в предметной области ВУЗ выделить подклассы «Сотрудник» и «Студент», то следует использовать знак неисключительной категории, так как студент одновременно может быть и сотрудником института.

На рис. 4.38 приведен пример иерархии обобщения в нотации IE. Он получен путем преобразования модели, представленной на рис. 4.33 из нотации IDEF1X в нотацию IE.

Рис. 4.38. **Пример иерархии обобщения в нотации IE**

В палитре инструментов для нотации IE присутствует только символ эксклюзивной категории. При преобразовании модели из нотации IDEF1X в нотацию IE всегда используется символ эксклюзивной категории. Если есть необходимость использовать знак неисключительной категории, то надо в окне ***Subtype Relationship*** выбрать радиокнопку ***Inclusive*** (рис. 4.39).

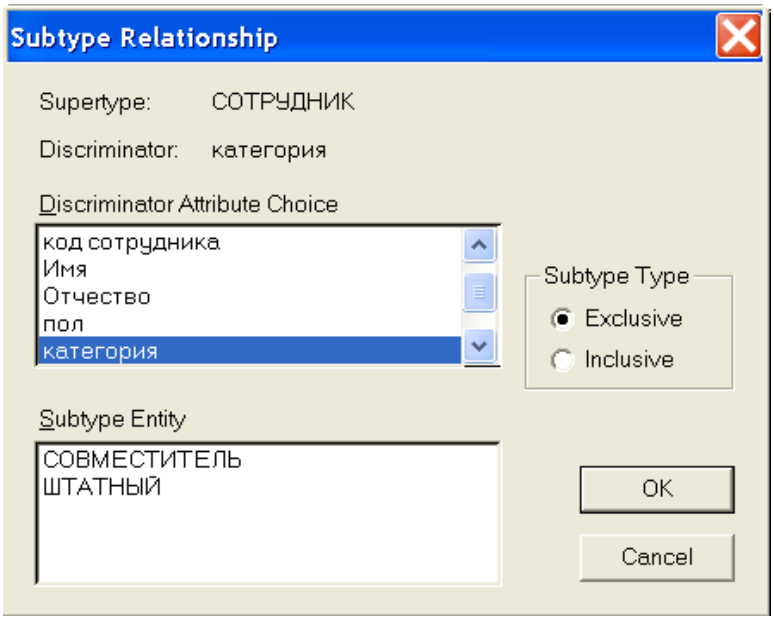


Рис. 4.39. **Вид окна Subtype Relationship для нотации IE**

4.7. Задание связей между сущностями

4.7.1. 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».

В нотации IDEF1X для этих типов связей используются следующие условные обозначения:

- Идентифицирующая связь «один-ко-многим» ();
- Неидентифицирующая связь «один-ко-многим» ();
- Связь «многие-ко-многим» ().

Для того чтобы задать связь, надо 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.

Для того чтобы отредактировать свойства связи, следует, установив указатель мыши на линии связи, щелкнуть правой кнопкой мыши и в появившемся контекстном меню (рис. 4.40) выбрать позицию ***Relationship Properties***.

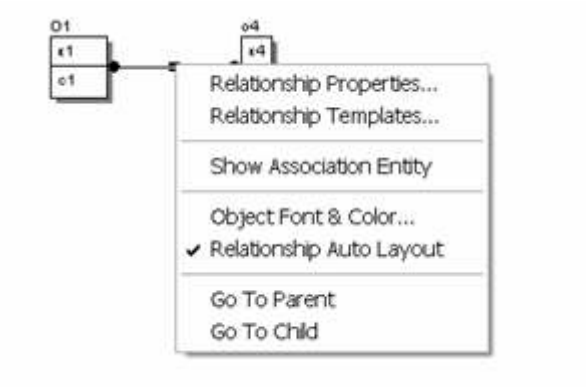


Рис. 4.40. **Контекстное меню для связи**

В появившемся окне (рис. 4.41) 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.

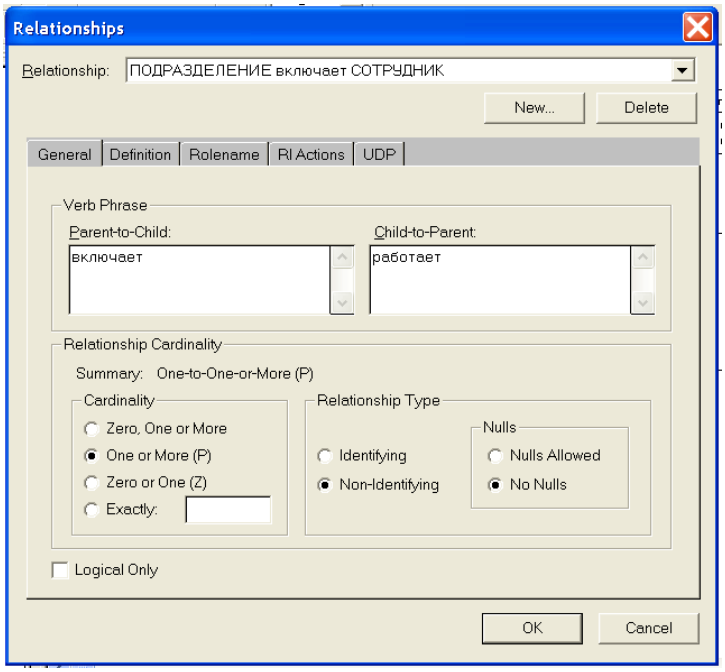


Рис. 4.41. **Окно Relationships**

При задании типа связи (RelationshipType) в нотации ERWin/IDEF1X 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 неидентирующей связи выбрана опция ***Null Allowed***, то на соответствующем конце связи используется символ ромбика.

Для обозначения кардинального числа (Cardinality) используются символы:

- P – один или много
- Z – ноль или один
- Выбранное при описании связи число – Exactly.

По умолчанию символ, определяющий кардинальное число, не отображается на схеме. Для его отображения следует в меню ***Format/Relationship Display*** отметить соответствующую опцию (рис. 4.42).

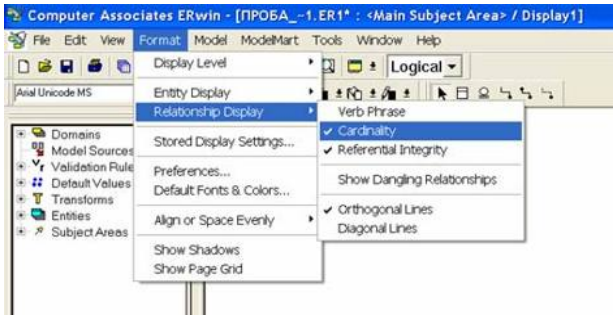


Рис. 4.42. **Задание параметров отображения связей**

Связь 1:M

На рис. 4.41 изображено описание связи между сущностями ПОДРАЗДЕЛЕНИЕ и СОТРУДНИК. В подразделении должен работать хотя бы один сотрудник (Cardinality – One or More (P)); идентификатор сотрудника не зависит от идентификации подразделений (тип связи – Non-Identifying); сотрудник обязательно приписан к какому-либо подразделению (Nulls/No Nulls). Соответствующий фрагмент логической модели в нотации IDEF1X приведен на рис. 4.43.

Рис. 4.43. **Пример неидентифицирующей связи с обязательным классом членства**

Если сотрудник может быть не приписан ни к какому отделу, то в окне Relationships (рис. 4.41) следует выбрать радиокнопку ***Null Allowed.*** В схеме модели на конце линии связи появится ромбик (рис. 4.44).

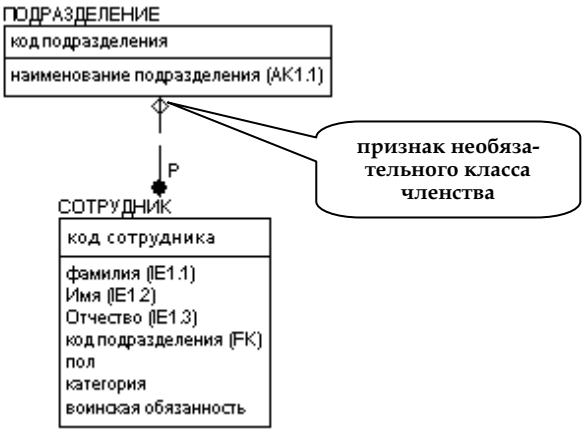


Рис. 4.44. **Пример неидентифицирующей связи с необязательным классом членства**

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 (рис. 4.45).

Рис. 4.45. **Пример идентифицирующей связи**

Различие при использовании идентифицирующей и неидентифицирующей связи заключается в том, что в первом случае ключ исходной сущности становится частью составного ключа зависимой сущности, а во втором – неключевым атрибутом. Миграция ключа исходной сущности в обоих случаях производится при установлении связи автоматически. И в том, и в другом случае ключ исходной сущности по отношению к зависимой сущности является внешним ключом (FK).

Связь M:M

Связь «многие-ко-многим» в реляционной модели не поддерживается и обычно реализуется путем создания связующей таблицы[1]. В ERWin при переходе от логической модели к физической система по умолчанию не преобразует связь M:M к виду, воспри-

[1] См.: Диго С.М. Базы данных: проектирование и использование.

нимаемому в реляционной системе. Для того чтобы указанное выше преобразование выполнялось автоматически, надо в меню ***Model*** выбрать позицию ***Model Properties*** и в появившемся окне (рис. 4.46) отметить позицию **Many-to-Many Relationships with Association Table**. Если такое преобразование не задать, то связующая таблица создана не будет.

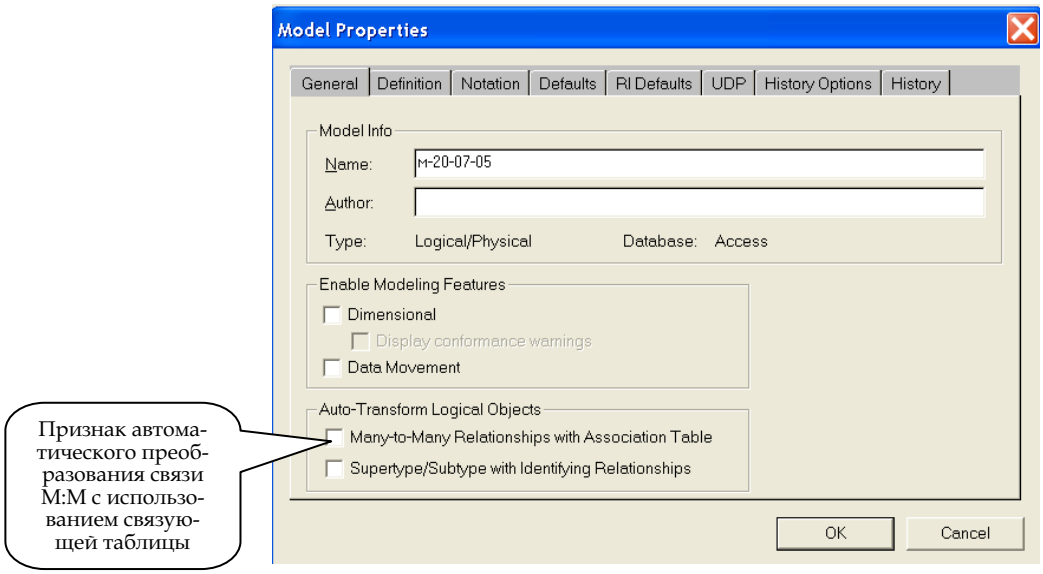


Рис. 4.46. **Задание признака автоматического преобразования связи M:M с использованием связующей таблицы**

На рис. 4.47 приведен пример связи M:M между сущностями ПРЕПОДАВАТЕЛЬ и ПРЕДМЕТ.

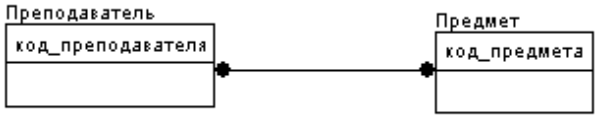


Рис. 4.47. **Пример связи M:M (логический уровень)**

4.7.2. Пример логической модели в нотации IDEF1X

На рис. 4.48 изображен фрагмент ER-модели в нотации IDEF1X, включающий все допустимые типы связей 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 ЗАГРАНПАСПОРТОМ предполагалось, что сотрудник может иметь только один загранпаспорт либо не иметь его вообще.

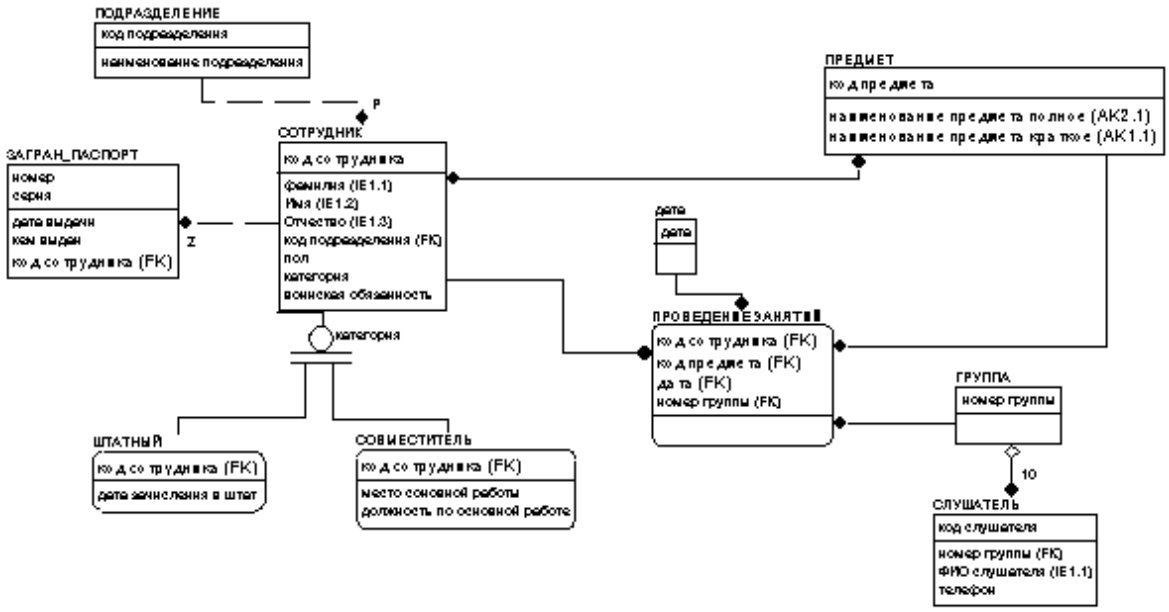


Рис. 4.48. **Пример логической модели в нотации IDEF1X**

4.7.3. Задание имен связей

В ERWin можно задавать, а можно и не задавать имена связи (***Verb Phrase***). Для того чтобы задать имя связи, надо войти в редактор связей (окно **Relationship**) и на вкладке ***General*** в блоке ***Verb Phrase*** в окнах **Parent-to-Child** и **Child-to-Parent** соответственно задать имя связи в прямом или обратном направлении соответственно (рис. 4.49).

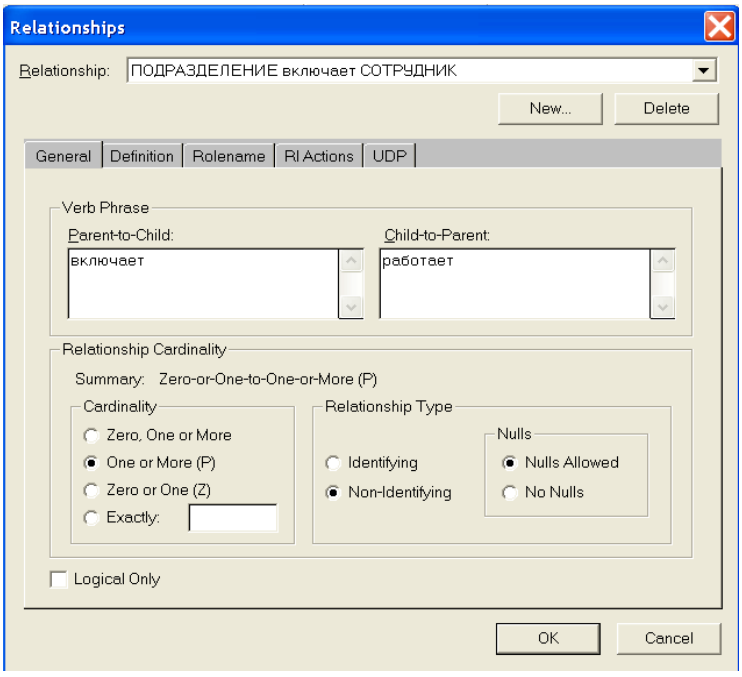


Рис. 4.49. **Задание имени связи**

Заданные при описании связи имена могут отражаться, а могут и не отображаться на экране. Для управления этим процессом можно воспользоваться позицией меню ***Format/Relationship Display*** и отметить позицию ***Verb Phrase*** (рис. 4.50).

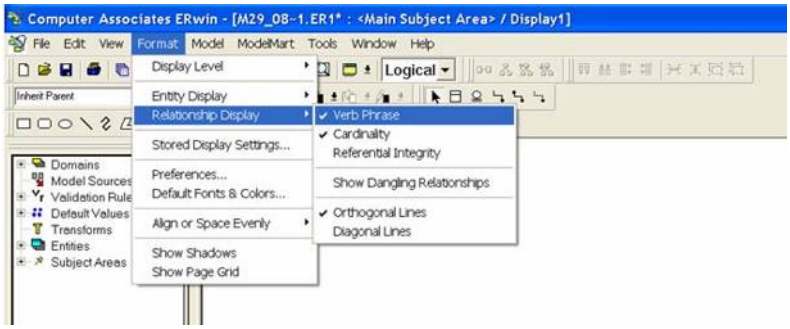


Рис. 4.50. **Обеспечение вывода имени связи в модели**

На рис. 4.51 представлен фрагмент ER-модели с отображенными на ней именами связей.

Рис. 4.51. **Фрагмент ER-модели с отображенными именами связи**

4.7.4. Задание нескольких связей между парой сущностей

В ERwin возможно задание нескольких связей между парой объектов (рис. 4.52). По умолчанию в этом случае на схеме в подчиненной сущности появляется только один внешний ключ.

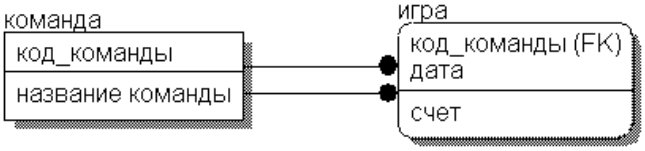


Рис. 4.52. **Задание нескольких связей между парой объектов**

Для того чтобы появились оба ключа, при описании связи на вкладке ***Rolename*** (рис. 4.53) следует указать ролевые имена (***Rolename***).

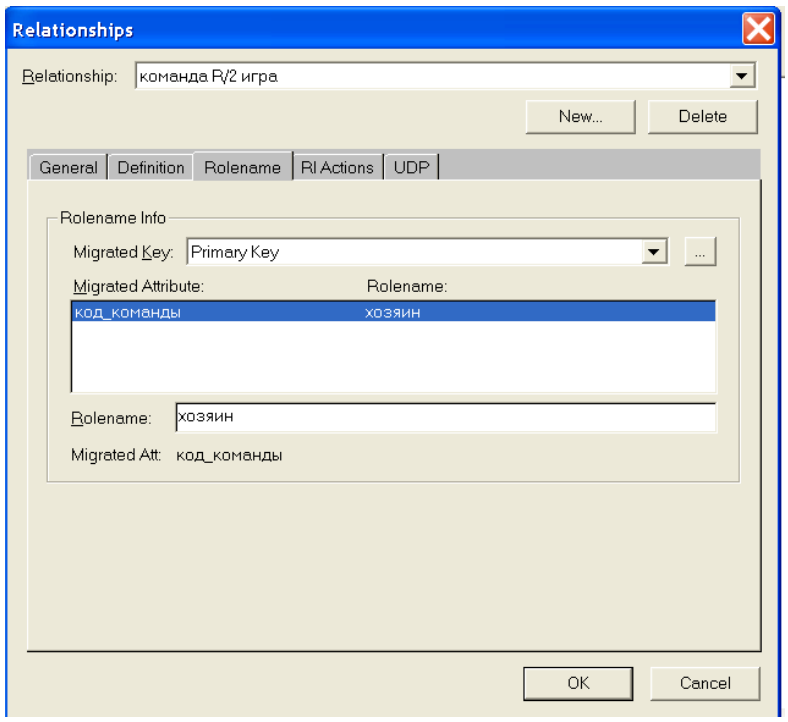


Рис. 4.53. **Задание ролевого имени**

На рис. 4.54 представлен фрагмент модели после задания ролевых имен.

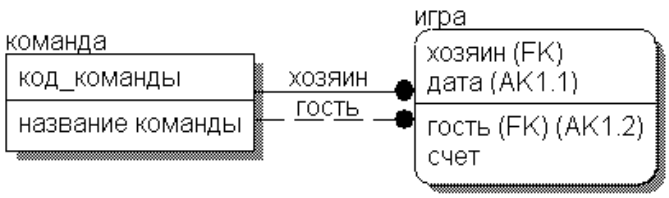


Рис. 4.54. **Вид модели при задании нескольких связей между парой объектов и задании ролевых имен связи**

4.7.5. Вид модели в нотации IE (Information Engineering)

Модель в нотации IDEF1X, представленная на рис. 4.48, в нотации IE имеет вид, представленный на рис. 4.55.

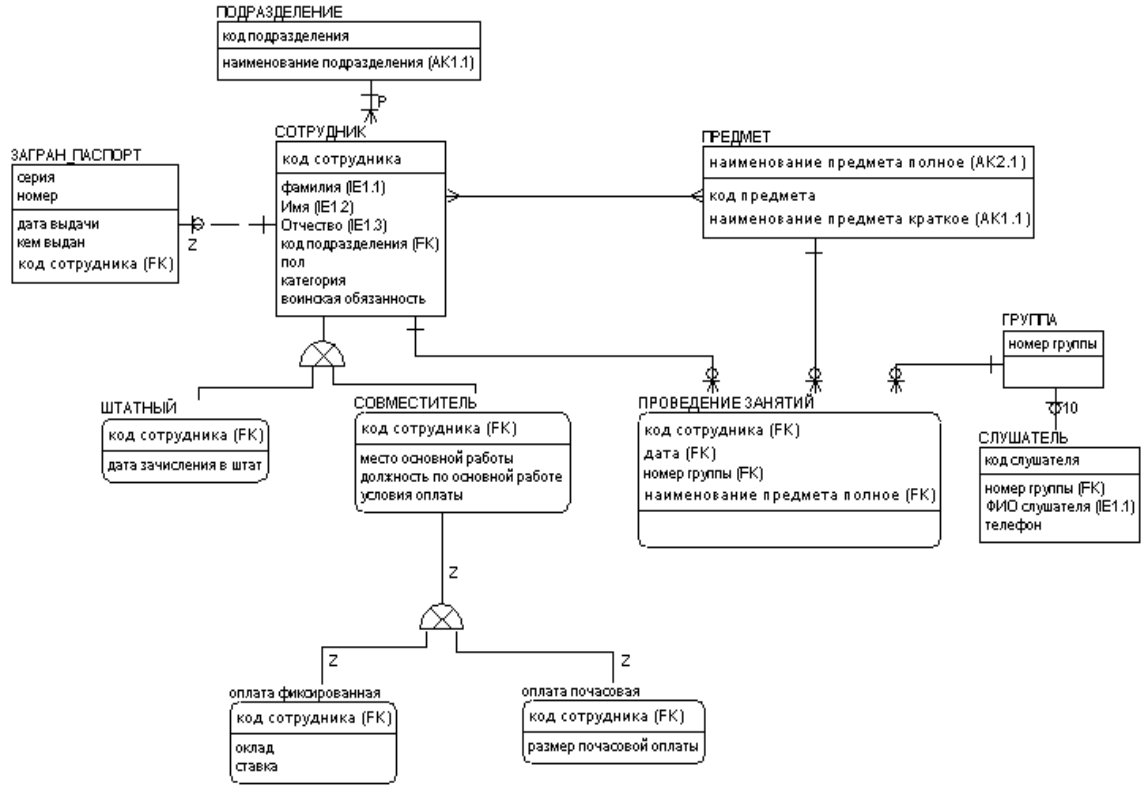
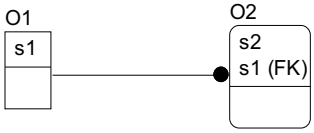
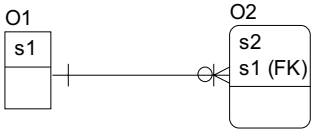
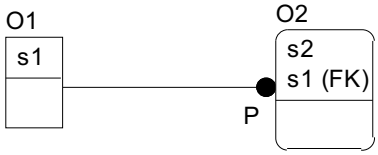
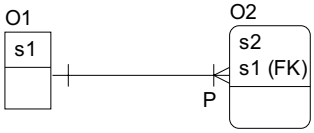
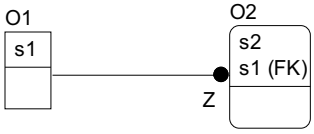
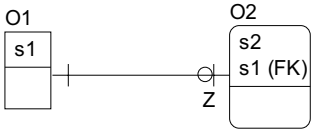


Рис. 4.55. **Вид модели в нотации IE**

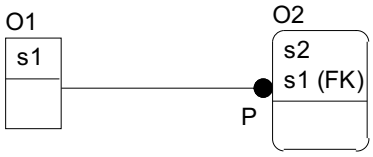
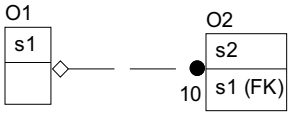
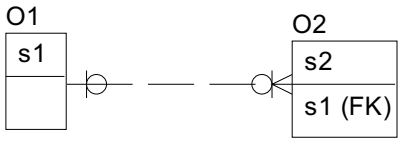
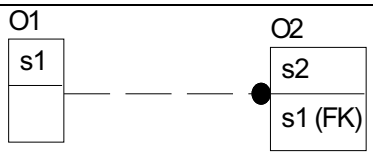
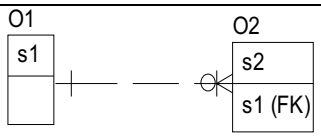
Идентифицирующая связь, так же как и в нотации IDEF1X, изображается в нотации IE сплошной линией, а неидентифицирующая – пунктирной.

Таблица 4.1

Соответствия между условными обозначениями нотаций IDEF1X и IE

Вид связи	Изображение в IDEF1X	Изображение в IE
Идентифицирующая связь Zero, One or Many		
Идентифицирующая связь One or Many		
Идентифицирующая связь Zero, One		

Окончание табл. 4.1

Вид связи	Изображение в IDEF1X	Изображение в IE
Идентифицирующая связь Exactly		
Неидентифицирующая связь Zero, One or Many, Nulls allowed		
Неидентифицирующая связь Zero, One or Many, No nulls		

4.8. 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. 4.56).

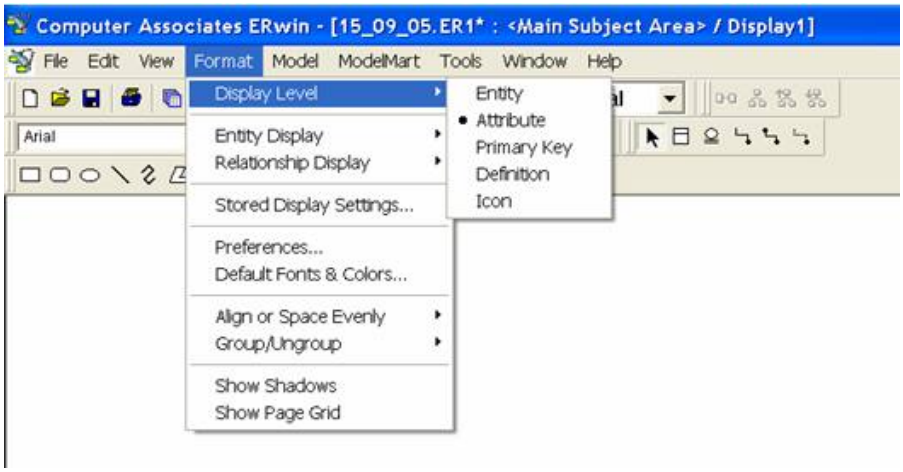


Рис. 4.56. **Выбор уровня отображения модели (Display Level)**

Кроме того, для изменения уровня просмотра модели можно воспользоваться панелью элементов (рис. 4.57).

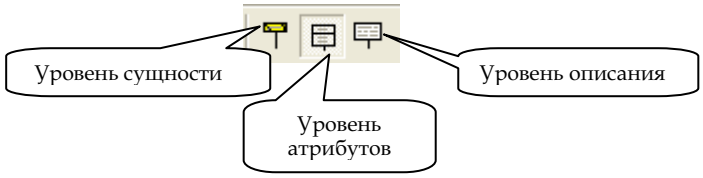


Рис. 4.57. **Элементы стандартной панели, используемые при изменении уровня отображения модели**

Во всех предшествующих примерах при изображении модели использовался уровень Attribute. На рис. 4.58 показан вид модели при выборе уровня Entity.

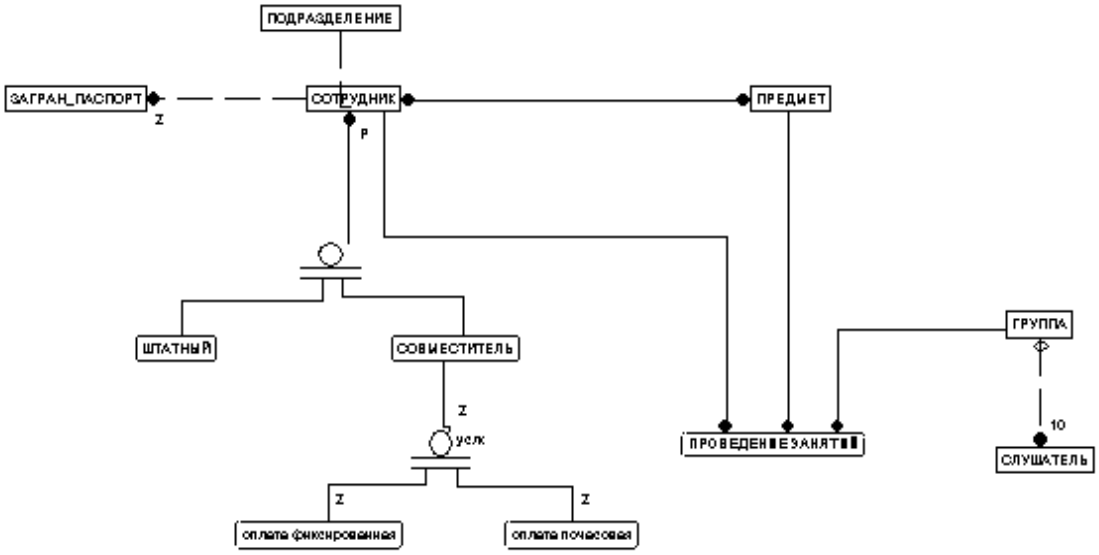


Рис. 4.58. **Отображение ER-модели – уровень сущности**

При выборе уровня Definition внутри прямоугольника, отображающего сущность, пишется то определение, которое было в окно **Definition** при описании сущности (рис. 4.59).

Рис. 4.59. **Отображение ER-модели – уровень описания**

При использовании уровня иконок в прямоугольнике, соответствующем сущности, кроме названия сущности изображается специально подобранная иконка (рис. 4.60), что может повысить наглядность модели.

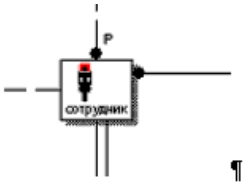


Рис. 4.60. **Отображение ER-модели – уровень иконок**

4.9. Ограничения целостности

При построении ER-модели в ERWin можно задавать ограничения целостности. Способ задания ограничения целостности зависит от вида ограничения.

4.9.1. Ограничения на значения атрибутов

Некоторые из ограничений целостности можно задать при определении атрибутов. На вкладке ***Datatype*** окна **Attributes** (рис. 4.61) можно отметить свойство ***Required***, что будет означать, что при вводе информации в базу данных для соответствующего поля должно быть обязательно введено значение. Для тех атрибутов, которые выбраны в качестве первичного ключа, это свойство является неактивным, так как свойство обязательности и так (по определению ключа) присуще элементам ключа.

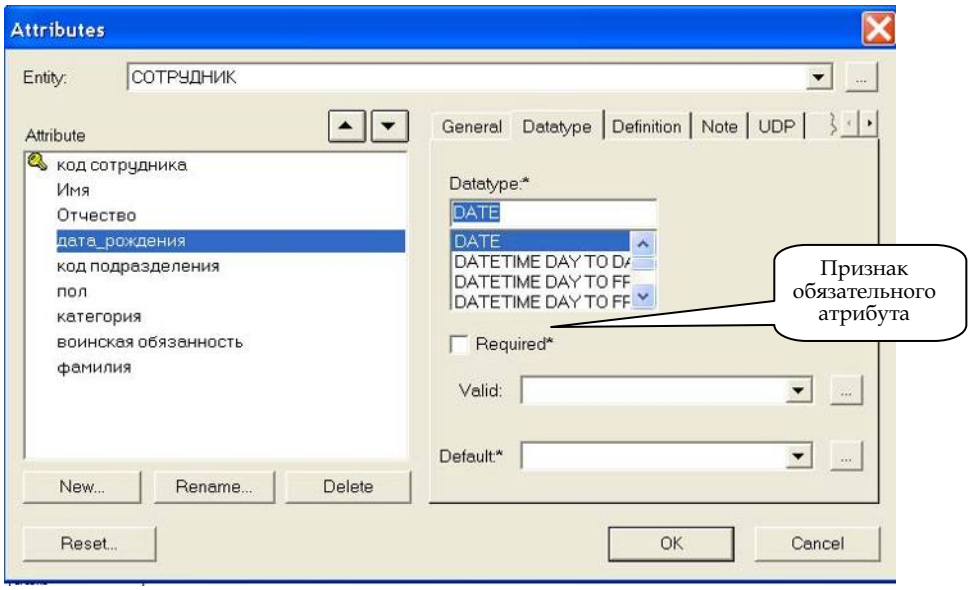


Рис. 4.61. **Окно описания атрибутов. Вкладка Datatype**

Для того чтобы задать ограничение на значение атрибута, надо нажать на кнопку ... рядом с окошком для задания значения параметра Valid. В появившемся далее окне Validation Rules (рис. 4.62) можно задать новые ограничения целостности.

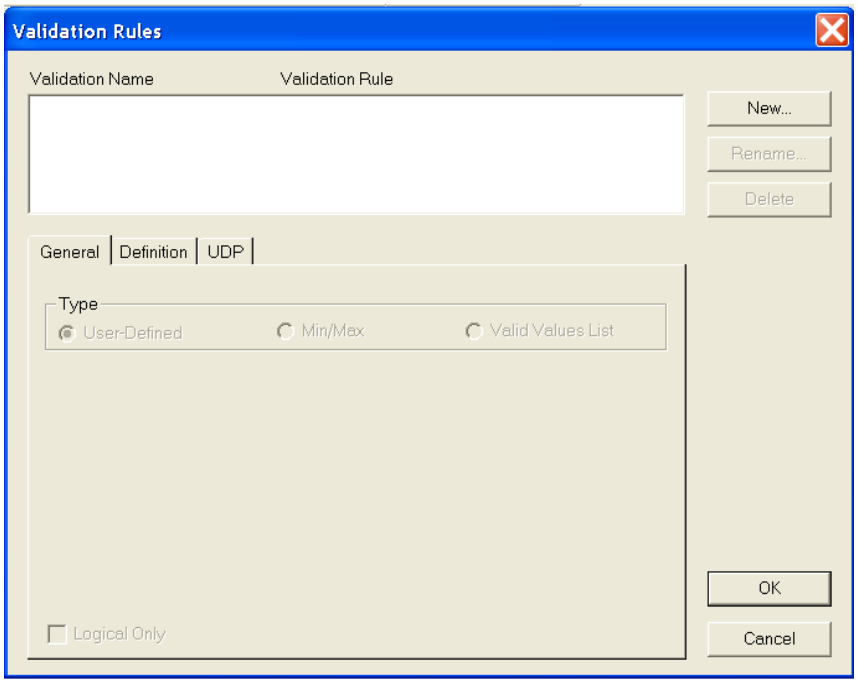


Рис. 4.62. **Задание ограничений целостности на значения атрибута. Окно Validation Rules**

Для задания нового ограничения целостности надо нажать на кнопку ***New*** и в появившемся окне **New Validation Rule** (рис. 4.63) задать имя ограничения.

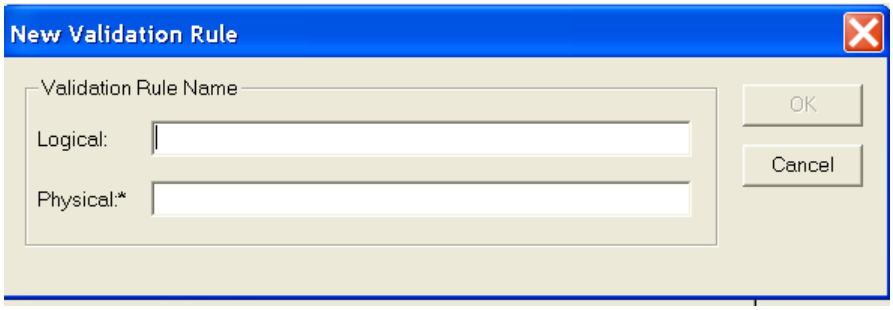


Рис. 4.63. **Задание имени ограничения целостности**

В окне **Validation Rules** (рис. 4.64) можно задать список допустимых значений для выбранного атрибута (***Valid Values List***).

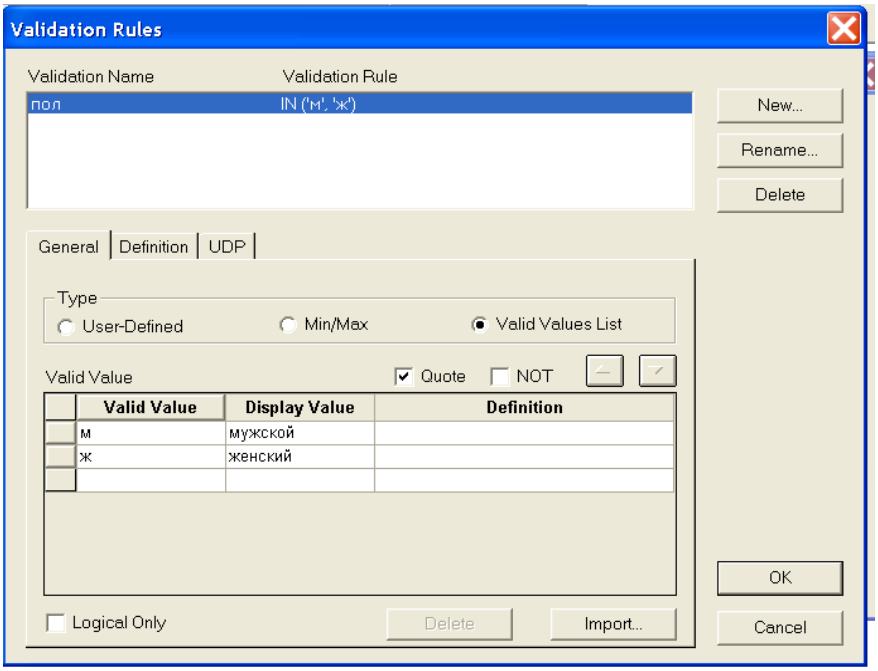


Рис. 4.64. **Задание списка допустимых значений атрибутов**

В рассматриваемом примере был задан допустимый список значений для атрибута «пол». В окне **Attributes** заданное ограничение целостности выглядит так, как показано на рис. 4.65.

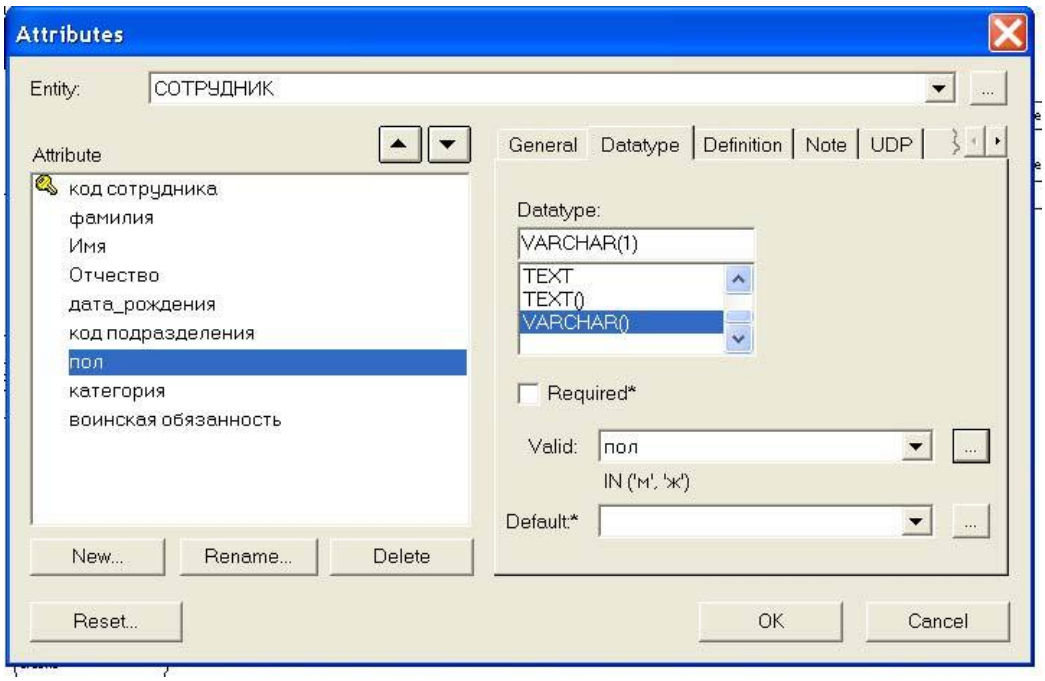


Рис. 4.65. **Отображение ограничения целостности для атрибута «пол» в окне Attributes**

Кроме задания списка значений для атрибута можно определить максимальное и минимальное значение. Для этого в окне при описании ограничения целостности следует выбрать тип ограничения ***Min/Max*** (рис. 4.66).

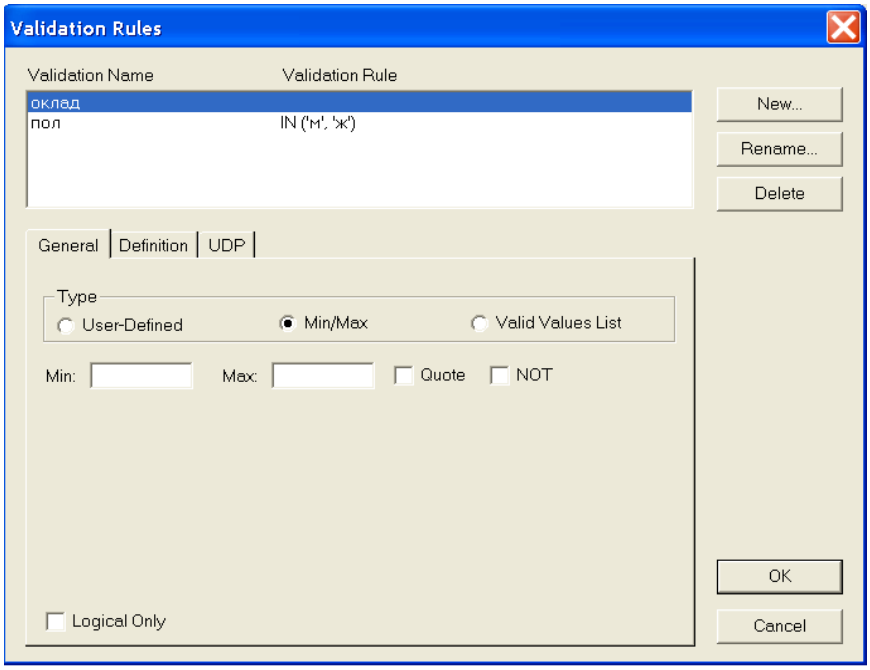


Рис. 4.66. **Задание диапазона значений атрибута**

В рассматриваемом примере для атрибута «оклад» был задан диапазон значений от 8 000 до 40 000. В окне **Attributes** заданное ограничение целостности выглядит так, как показано на рис. 4.67.

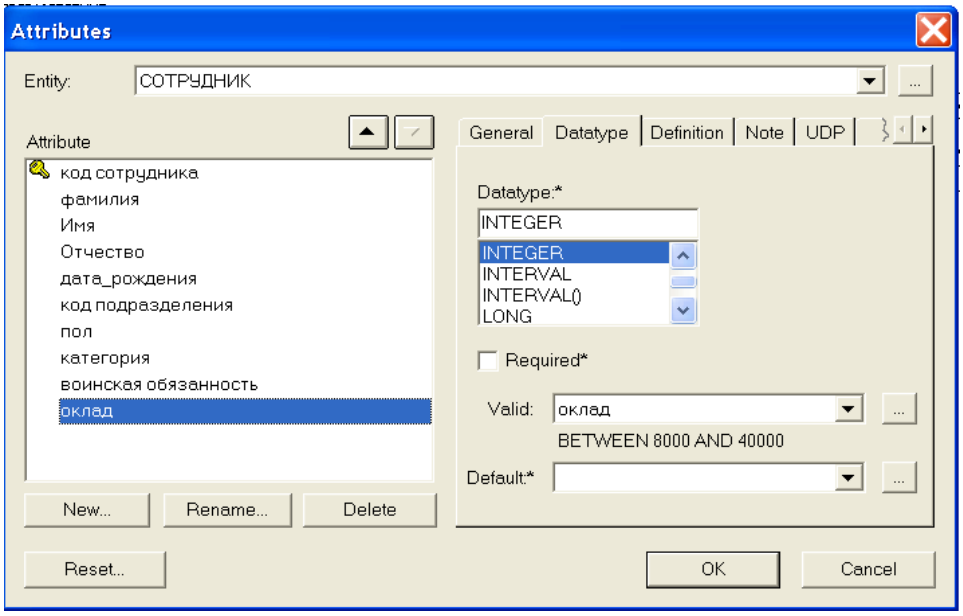


Рис. 4.67. **Вид окна описания атрибутов после задания ограничения целостности на диапазон значений атрибута «оклад»**

При задании ограничений целостности можно воспользоваться и возможностью задать любое допустимое выражение, выбрав тип ограничения ***User-Defined*** (рис. 4.68). В приведенном примере задано выражение, определяющее возраст сотрудника.

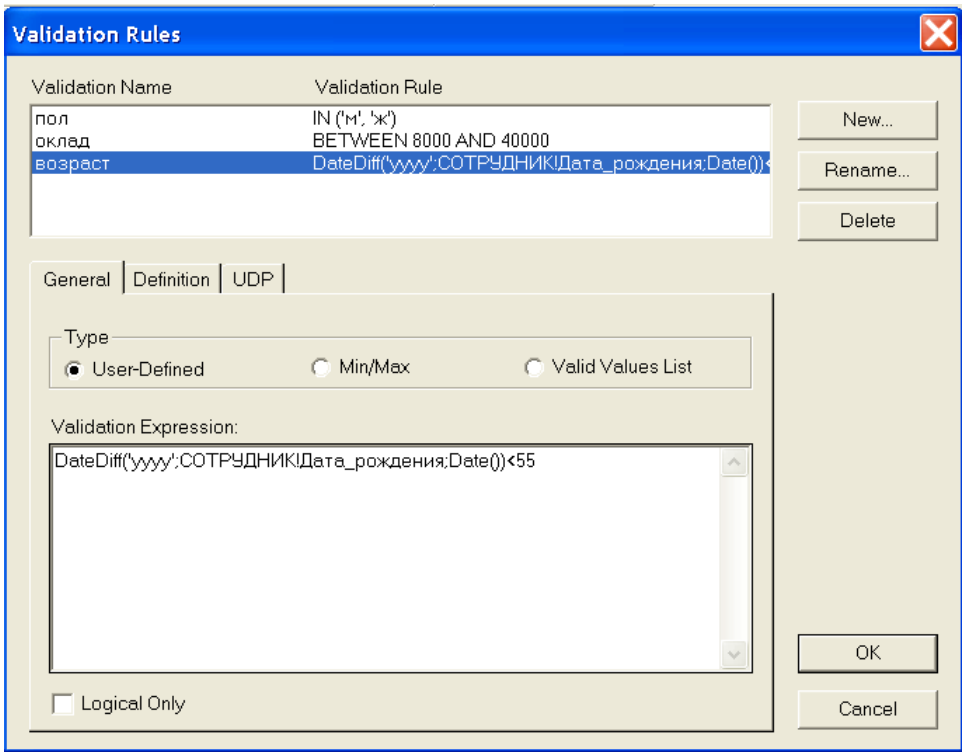


Рис. 4.68. **Задание определенного пользователем ограничения целостности**

В окно задания ограничений целостности можно попасть и выбрав позиции меню ***Model/Validation Rules*** (рис. 4.69). Если ранее уже были заданы какие-то ограничения целостности, то в окне **Validation Rules** будет выводиться весь список этих ограничений.

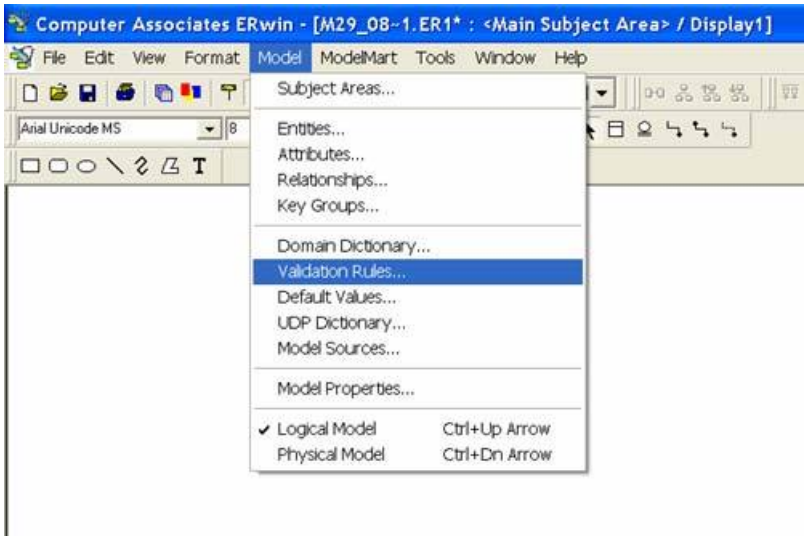


Рис. 4.69. **Меню Model/Validation Rules**

4.9.2. Ограничения целостности связи

При описании связи можно задать ограничения целостности связи. Для этого следует воспользоваться вкладкой **RI Actions** (рис. 4.70) в окне редактора связей.

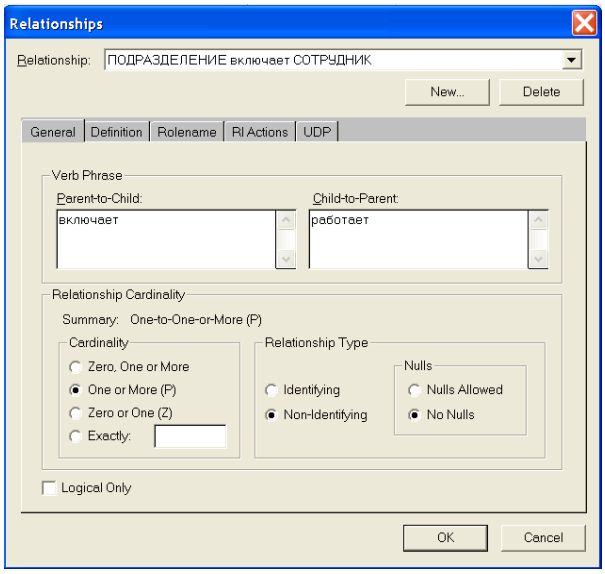


Рис. 4.70. **Выбор вкладки RI Actions для задания ограничений целостности связи**

Вид окна Relationships для этой вкладки показан на рис. 4.71. Для каждой связи можно задать действия, которые будут выполняться при удалении (Delete), вставке (Insert) и удалении (Update) как порожденной (Child), так и родительской (Parent) сущности.

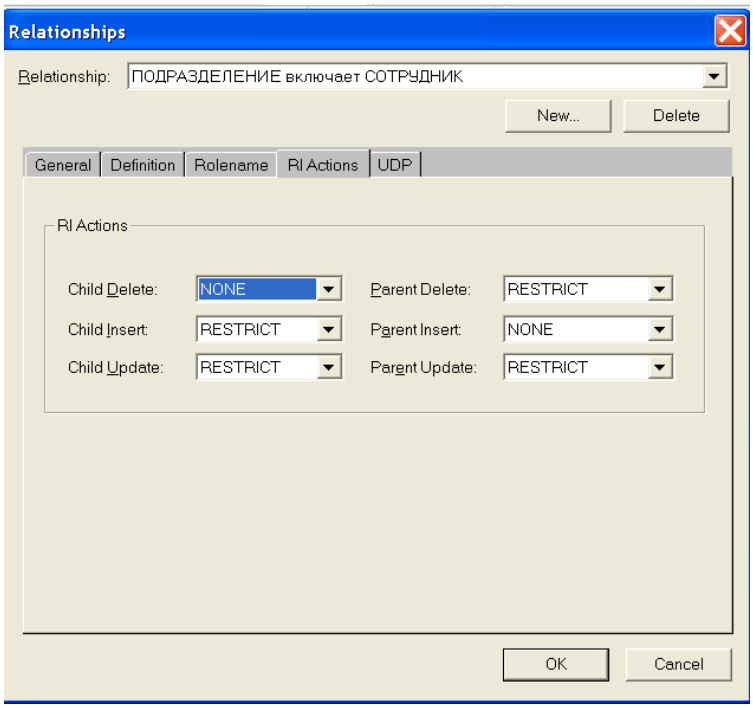


Рис. 4.71. **Окно задания ограничения целостности связи**

Для каждой из корректирующих операций можно выбрать следующее действие:

- NONE – действие не оказывает влияние 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.

Набор возможных значений и значения, выбранные системой по умолчанию, будут зависеть от характеристик связи, для которой задаются ограничения целостности.

Как видно из рис. 4.48, связь «ПОДРАЗДЕЛЕНИЕ – СОТРУДНИК» была задана как неидентифицирующая с обязательным классом членства объектов в связи (Not Null). В этом случае каждый из списков, определяющих действия, выполняемые при корректировке, имеет четыре возможных значения: NONE (никакой), RESTRICT (ограничивать), CASCADE (каскад), SET DEFAULT (значение по умолчанию).

На рис. 4.71 показаны значения ***RI Action***s для связи «ПОДРАЗДЕЛЕНИЕ – СОТРУДНИК», заданные системой по умолчанию.

Если изменить характер связи и определить класс членства как необязательный (Nulls Allowed), то выбранные по умолчанию значения для ограничений целостности будут иные (рис. 4.72).

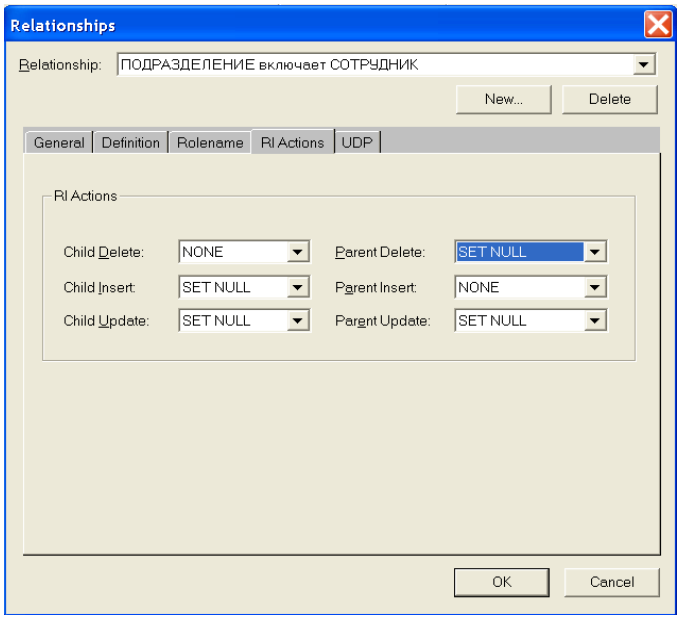


Рис. 4.72. **Значения по умолчанию для неидентифицирующей связи с необязательным классом членства объектов**

Выбор значения SET NULL для операции удаления родительской сущности в этом случае означает, что всем сотрудникам отдела, который удаляется (Parent Delete), в качестве значения поля ПОДРАЗДЕЛЕНИЕ будет присвоено значение NULL.

Для случая изменения значения поля КОДА ПОДРАЗДЕЛЕНИЯ (Parent Update) в нашем случае следует выбрать CASCADE, что будет вызывать каскадное изменение соответствующих значений внешнего ключа для записей всех сотрудников, работающих в данном отделе.

Рассмотрим другую связь, а именно «СОТРУДНИК – ЗАГРАНПАСПОРТ» (рис. 73).

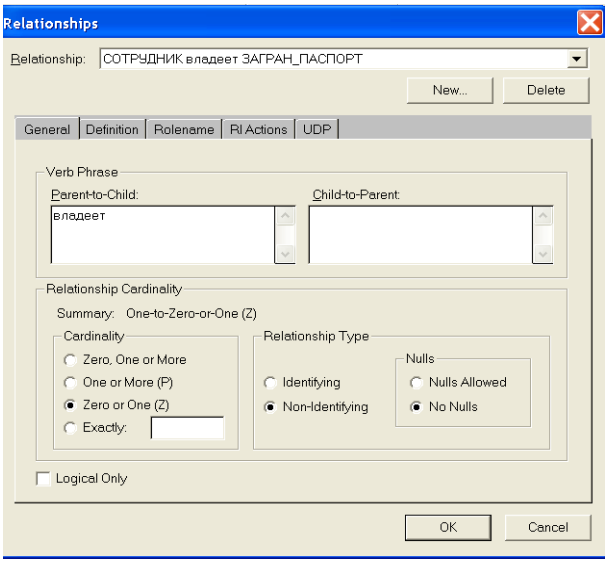


Рис. 4.73. **Описание связи «СОТРУДНИК – ЗАГРАНПАСПОРТ»**

Как мы видим, и в этом примере связь была задана как неидентифицирующая с обязательным классом членства объектов в связи (Not Null). Но в рассматриваемом случае 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».

Как было отмечено выше, выбор режима действий при выполнении корректирующих операций будет зависеть от типа связи между сущностями, а также от особенностей предметной области. В табл. 4.2 приведены возможные режимы для каждого вида связи. Значения по умолчанию выделены полужирным курсивом с подчеркиванием.

Таблица 4.2

Ограничения целостности связи

Роль сущности в связи	**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

Окончание табл. 4.2.

Роль сущности в связи	Идентифицирующая связь	Неидентифицирующая связь (Nulls Allowed)	Неидентифицирующая связь (No Nulls)	Категориальная связь
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***
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 можно изменить, выбрав позицию меню ***Model/Validation Rules,*** и в появившемся окне (рис. 4.74) на вкладке **RI Default** задать нужные значения соответствующих параметров.

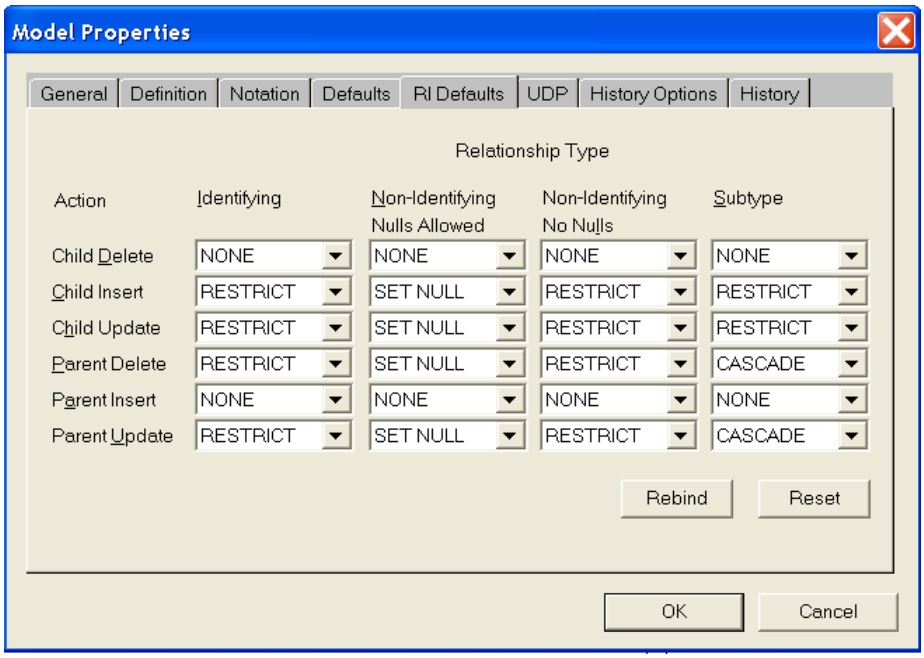


Рис. 4.74. **Окно Model Properties. Ограничения целостности, задаваемые по умолчанию (RI defaults)**

Обычно удаление зависимой записи (Child Delete) при любом из типов связи не требует дополнительных проверок на допустимость корректировки и не приводит к 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 произведена, 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 надо 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 объектом (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 две разные сущности, а отражает информацию об одном и том же объекте. Для обобщенного объекта желательно 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.

Для того чтобы заданные ограничения целостности отражались при изображении модели, можно в меню ***Format/Relationship Display/Referential Integrity*** отметить позицию ***Referential Integrity*** (рис. 4.75).

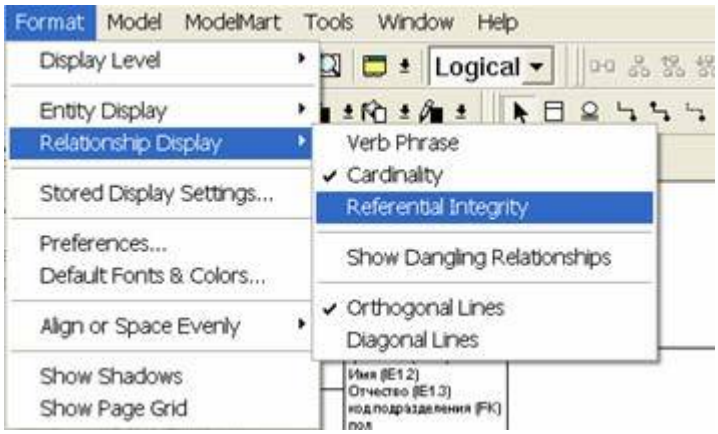


Рис. 4.75. **Меню Format/Relationship Display/Referential Integrity**

После такого выбора на схеме ER-модели на обоих концах каждой линии связи будут отображаться значения ограничений целостности связи. Они задаются двумя латинскими буквами, разделенными двоеточием (рис. 4.76). Первая буква обозначает опера-

цию, вторая – действия, предусмотренные при выполнении данной операции. Например, D:C означает, что при выполнении удаления экземпляра сущности (Delete) будет осуществляться каскадное удаление связанных сущностей (Cascade).

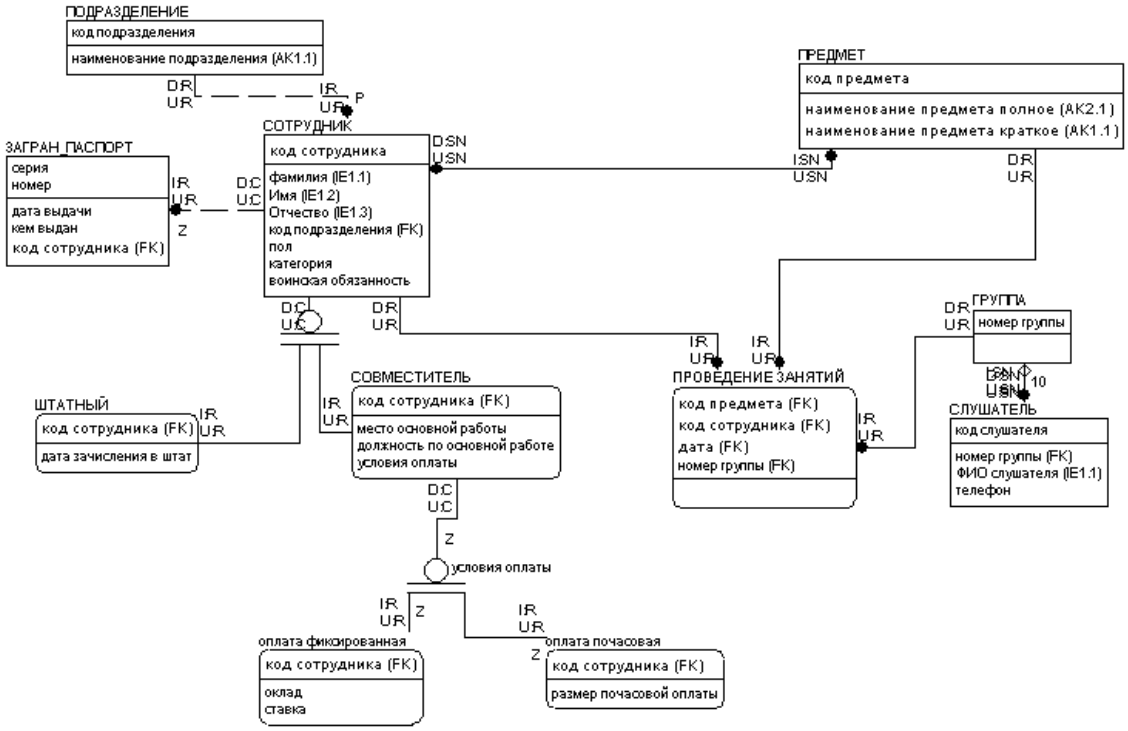


Рис. 4.76. **Вид ER-диаграммы с заданными ограничениями целостности**

4.9.3. 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 на таблице и выбрать во всплывающем меню пункт Triggers (рис. 4.77)

Рис. 4.77. **Контекстное меню для выбора позиции Triggers**

Так как не все СУБД поддерживают механизм триггеров, то соответствующая позиция меню присутствует не всегда: обычно она отсутствует для настольных СУБД и активна «корпоративных» систем.

4.10. Физическое моделирование

4.10.1. Выбор целевой СУБД

Как было отмечено ранее, в ERWin представление структуры базы данных для конкретной целевой СУБД называется ***физической моделью***. Если при создании модели был выбран шаблон представления «Logical/Physical», то для перехода к физической модели следует воспользоваться списком выбора, расположенным в правой части стандартной панели (рис. 4.78).

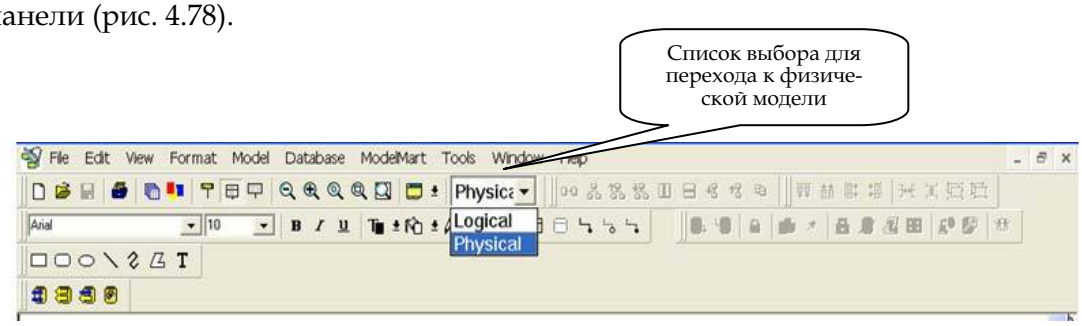


Рис. 4.78. **Переключение на физическую модель**

Перед созданием физической модели следует выбрать целевую СУБД. Для этого можно воспользоваться кнопкой  (***Select Target Server***) инструментального меню или позицией меню **Database/Choose Database**. В появившемся окне (рис. 4.79) надо выбрать желаемую СУБД и ее версию.

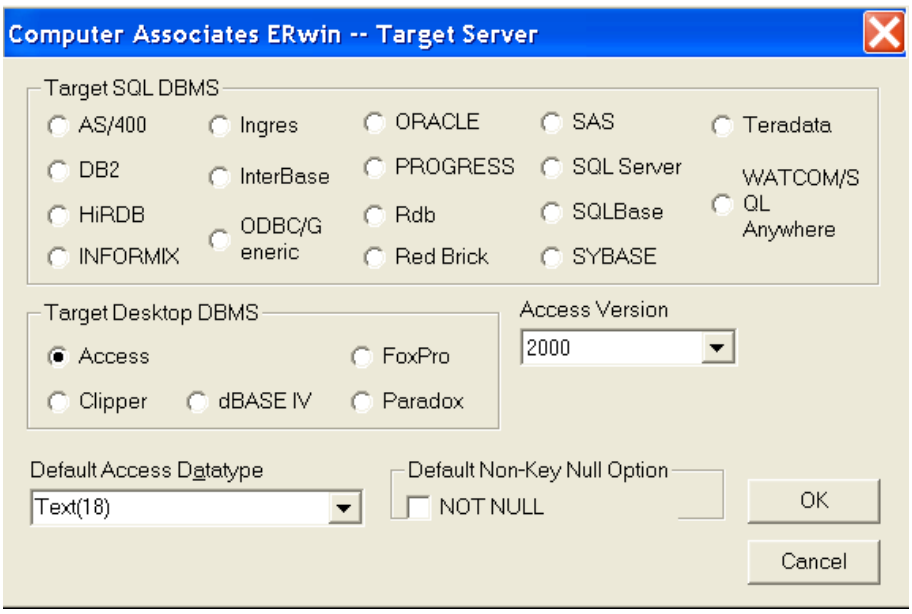


Рис. 4.79. **Выбор целевой СУБД**

Список целевых СУБД меняется от версии к версии. Причем в него не только вводятся новые СУБД, но и исключаются устаревшие. Если открывается ранее созданная ER-модель, в качестве целевой СУБД для которой была выбрана СУБД, которая теперь не поддерживается в ERWin, то система выдаст соответствующее предупреждение и предложит выбрать другую СУБД.

Вид окна **Target Server** будет несколько различаться в зависимости от выбранной СУБД.

Если в явном виде не была указана СУБД, то физическая модель строится для СУБД, выбираемой по умолчанию. Сменить целевую СУБД можно на любом этапе проектирования. Возможность использовать одну и ту же логическую модель для получения разных физических моделей является большим преимуществом CASE-технологий.

Если первоначально при создании концептуальной модели был выбран шаблон логической модели, то можно выбрать позицию меню ***Tools/Derive new model***. В появившемся окне **Derive new model** (рис. 4.80) надо выбрать нужный тип модели.

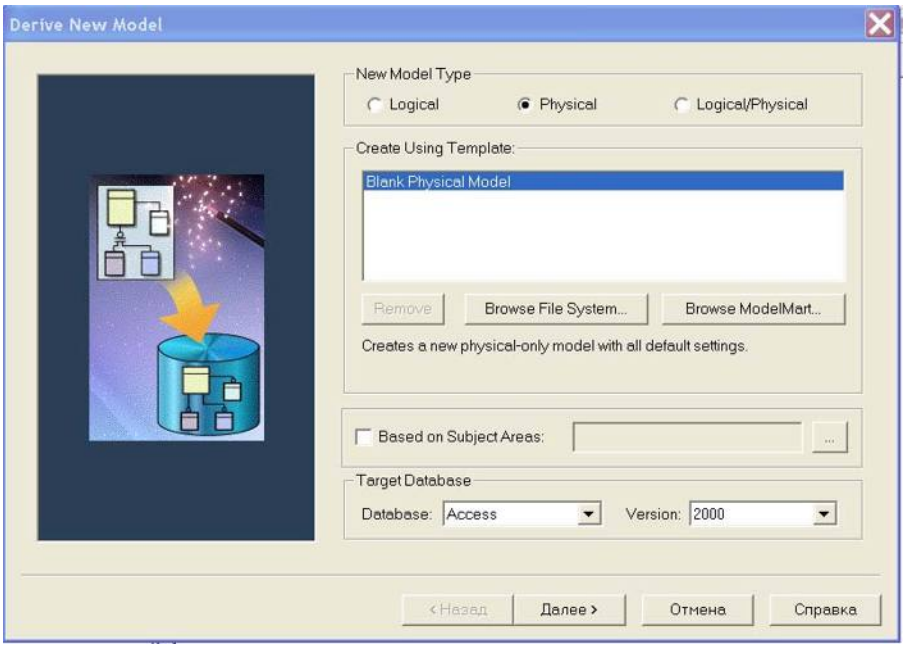


Рис. 4.80. **Выбор типа модели**

Если отдельно строятся логическая и физическая модели, то одной логической модели может быть поставлено в соответствие несколько физических моделей.

4.10.2. Нотации, используемые при построении физической модели

Также как и при изображении логической модели, для представления физической модели используется несколько нотаций (см. рис. 4.9), а именно: IDEF1X1X (Integration DEFinition for Information Modeling), IE (Information Engineering) и DM (Dimensional Modeling). Как мы видим, по сравнению с логическим моделированием, появилась дополнительная нотация – DM, о которой далее будет сказано особо.

Как указывалось ранее, для того чтобы иметь возможность изменить нотацию модели, следует выбрать позицию меню ***Model/Model Properties*** и перейти на закладку ***Notation.***

Вид панели инструментов (ERWin Toolbox) для физической модели в нотации IDEF1X1X представлен на рис. 4.81.

Рис. 4.81. **Вид панели инструментов (ERWin Toolbox) для физической модели в нотации IDEF1X1X**

Кнопка  соответствует таблице базы данных, кнопка  – преставлению (view). Кнопки  и  , в логической модели отражающие тип связи между сущностями, в физической модели фактически просто «визуализируют» связь, так как в реляционной модели связь передается включением полей связи в соответствующие таблицы. Кнопка  (view relationship) используется для связывания таблиц с представлениями. Понятие «преставление» (view) широко используется в языке SQL и обозначает виртуальную таблицу, описанную SQL-запросом. Для понимания ситуаций, когда следует создавать представления, и самого процесса создания представлений необходимо иметь соответствующие знания по SQL.

4.10.3. Сравнение логической и физической модели

На рис. 4.82 изображена физическая модель в нотации IDEF1X1X, полученная из логической модели, представленной на рис. 4.48.

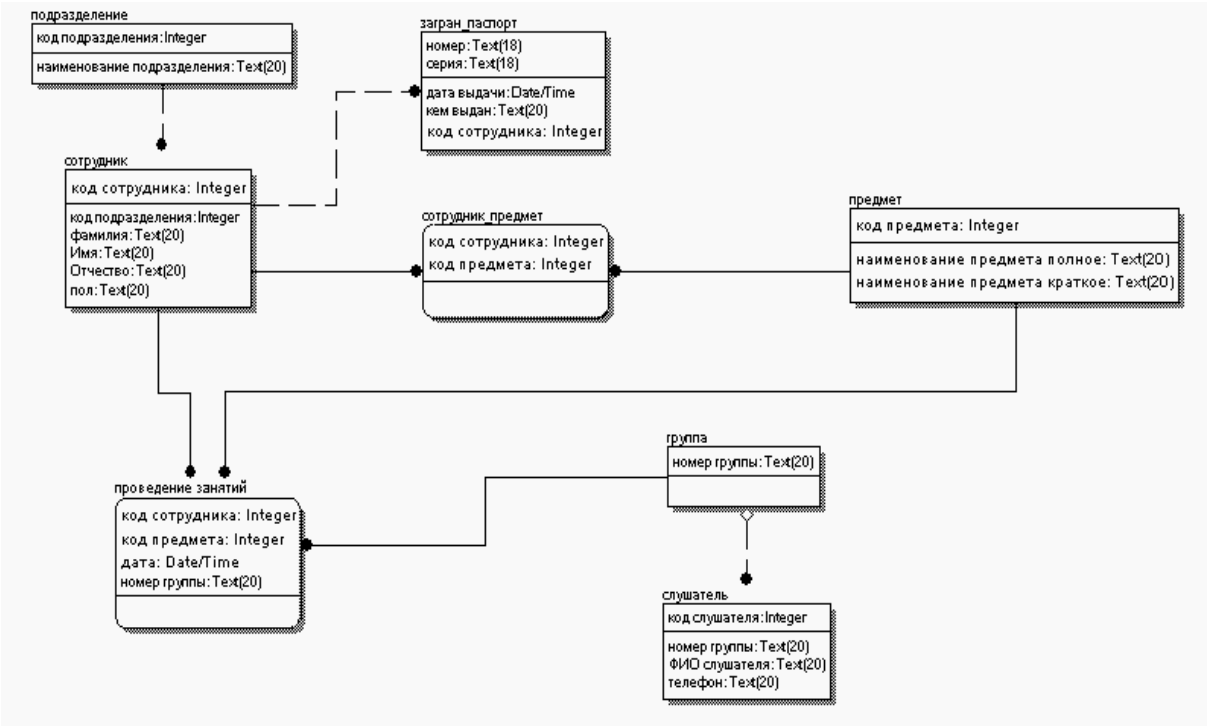


Рис. 4.82. **Физическая модель в нотации IDEF1X1X**

Сущности (Entity) в логической модели соответствует таблица (Table) в физической модели, атрибуту (Attribute) – соответствует колонка (Column). Для каждой колонки указывается тип данных и длина. Типы данных зависят от выбранной СУБД. Первичному и альтернативному ключу, а также инверсному входу в физической модели соответствуют индексы.

Сущность ДАТА была отмечена в логической модели как Logical Only. Поэтому на физической модели соответствующая ей таблица отсутствует.

В физической модели появилась дополнительная связующая таблица СОТРУДНИК_ПРЕДМЕТ, которая трансформирует связь M:M между этими сущностями. Преобразование связи M:M произошло автоматически, так как в ***Model Properties*** было установлено **Many-to-Many Relationships with Association Table**.

4.10.4. Преобразование связи «многие-ко-многим»

Как известно, связи M:M между таблицами баз данных в реляционной модели не поддерживаются. Если при создании связи M:M в логической модели в меню ***Model/Model Properties*** была отмечена позиция **Many-to-Many Relationships with Association Table**, то при переходе к физической модели вместо связи M:M будет создана связующая таблица, с которой таблицы, соответствующие сущностям, связанным отношением M:M, будут связаны обычной для реляционных моделей связью 1:M.

На рис. 4.83 изображен соответствующий фрагмент модели физического уровня, получившийся путем автоматического преобразования связи M:M (см. рис. 4.47) в связующую (ассоциативную) таблицу.

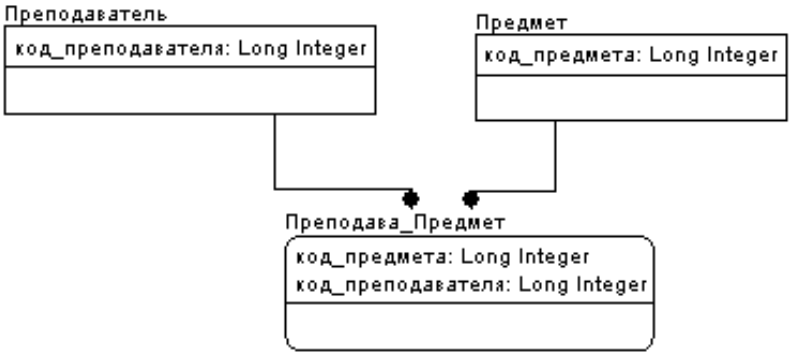


Рис. 4.83. **Создание связующей таблицы при преобразовании связи M:M**

Полученную таблицу можно переименовать, назвав ее, например, «Владение предметом».

Если при создании связи M:M в логической модели в меню ***Model/Model Properties*** позиция **Many-to-Many Relationships with Association Table** не была отмечена, то при переходе к физической модели связующая таблица создана не будет. Если все-таки надо, чтобы такая таблица была создана (а в подавляющем большинстве случаев это именно так), то можно воспользоваться возможностью трансформации модели. Для этого надо выделить связь и нажать на кнопку  (Many to Many Transform). И далее, следуя подсказкам системы, надо задать имя создаваемой связующей таблицы и имя трансформации.

В процессе создании ER-модели может возникнуть ситуация, что, уточняя и расширяя модель, в связующую таблицу будут введены дополнительные колонки (т.е. для связи будут заданы характеризующие ее свойства). Если не предпринять дополнительных действий, то в логической модели эти изменения не будут видны. Чтобы они стали видны и в логической модели, надо в «эксплоере» выбрать позицию **Transform/Delete/Resolve Transform** (рис. 4.84).

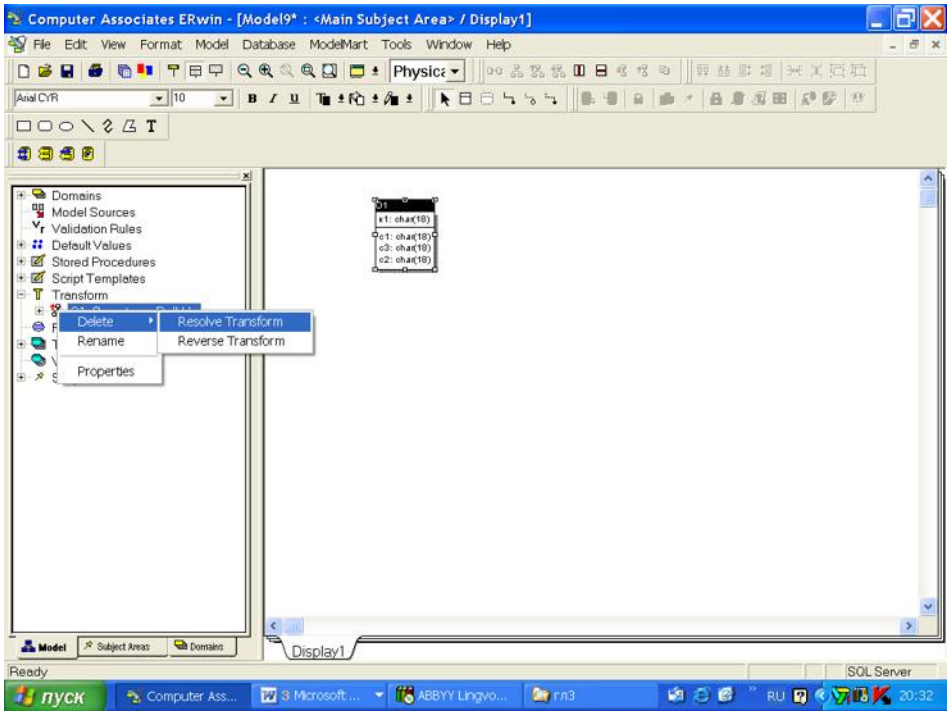


Рис. 4.84. **Выбор позиции Resolve Transform/Reverse Transform**

Несмотря на то, что в ERWin есть возможность использовать **Many to Many Transform** для каждой связи M:M, лучше задать признак автотрансформации в свойствах модели (см. рис. 4.46), так как обычно связь M:M в физической модели передается связующей таблицей. Если же связь M:M не отображается в физической модели, то скорее всего ее надо удалить из логической модели.

4.10.5. Отображение обобщенной сущности

В книге «Базы данных: проектирование и использование» при изложении алгоритма перехода от ER-модели к модели реляционной базы данных приведено несколько альтернативных вариантов решений при отображении обобщенного объекта в таблицы базы данных.

В ERWin при переходе от логической модели к физической возможность автоматического выбора альтернатив отсутствует. Для каждой из сущностей, как родовой (Supertype), так и видовых (Subtype), создается отдельная таблица (рис. 4.85).

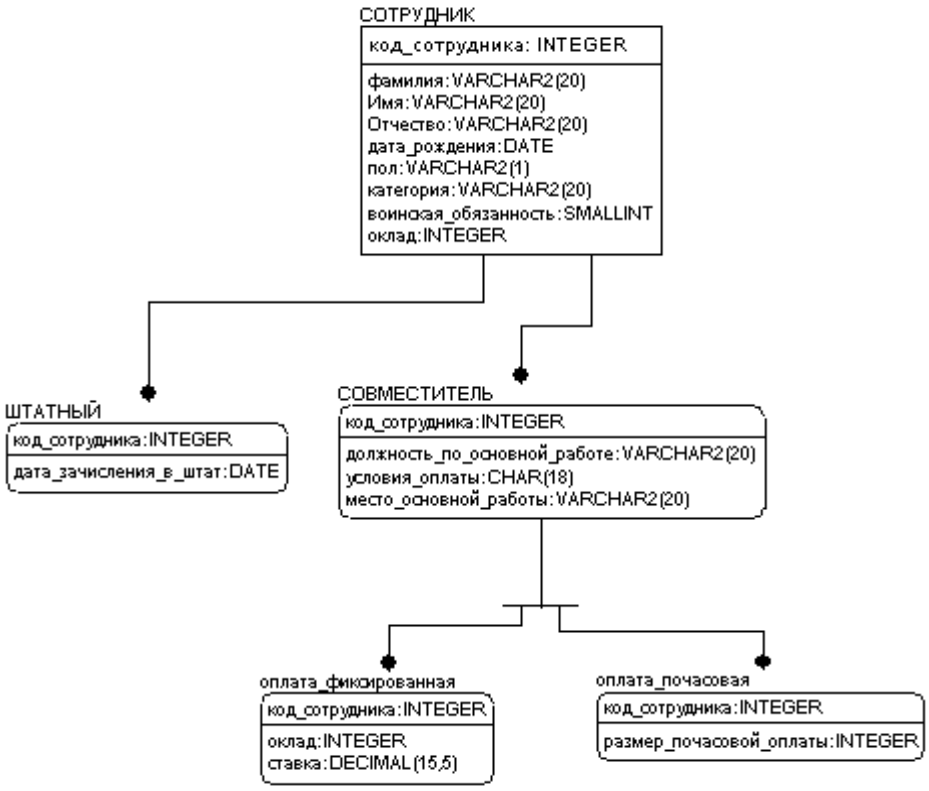
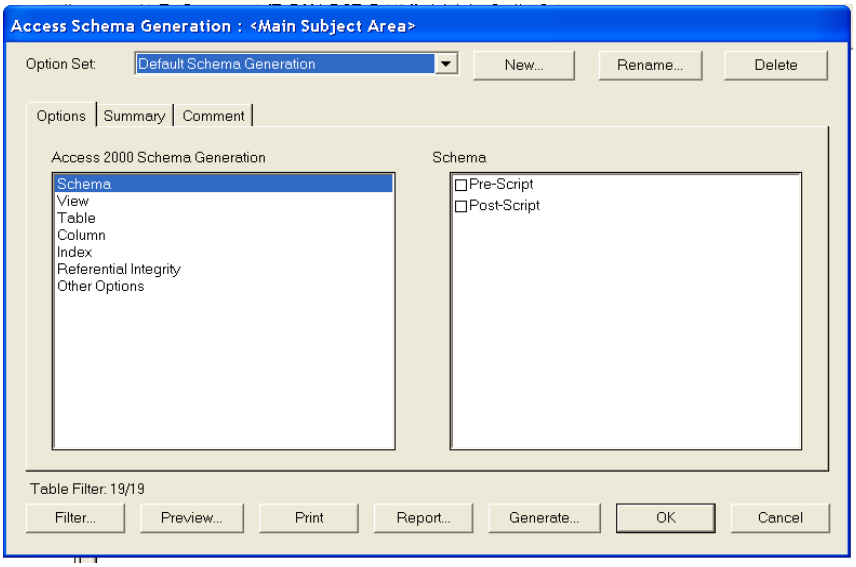


Рис. 4.85. **Отображение обобщенного объекта на физическом уровне**

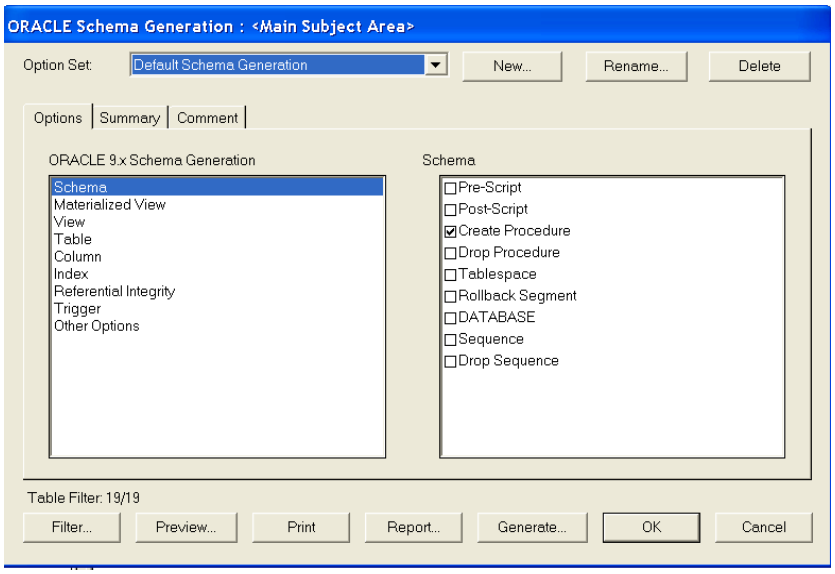
Но если такое решение не устраивает проектировщика, можно воспользоваться возможностью трансформирования модели.

4.10.6. Создание базы данных

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 нажать кнопку  на панели инструментов. В результате появится окно (рис. 4.86) генерации схемы в выбранной целевой СУБД. Вид окна зависит от выбранной целевой СУБД. При генерации схемы могут быть созданы: таблицы, триггеры, хранимые процедуры, индексы, ограничения целостности и другие объекты, поддерживаемые целевой СУБД.



а)



б)

Рис. 4.86. **Вид окна генерации схемы:**
а) целевая СУБД – Access; б) целевая СУБД Oracle

Кнопка Filter позволяет убрать объекты из схемы. Кнопка Preview позволяет просмотреть SGL-скрипт, создаваемый ERWin для генерации системного каталога СУБД. С помощью кнопки Print можно вывести на печать созданный SGL-скрипт. Кнопка Report сохранит тот же скрипт в текстовом файле. Кнопка Generate запускает процесс генерации схемы.

Если ER-модель будет использоваться для создания новой базы данных, то надо сначала создать пустую базу данных в среде выбранной целевой СУБД.

Контрольные вопросы

1. Какие нотации используются в AllFusion ERwin Data Modeler при изображении ER-модели?
2. Какие виды сущностей присутствуют в нотации IDEF1X?
3. Какие виды сущностей присутствуют в нотации IE?
4. Какие виды связей присутствуют в нотации IDEF1X?
5. Какие типы связей присутствуют в нотации IE?
6. В чем заключается разница между физической и логической моделью?
7. Что в ERWin называется физической моделью?
8. Как можно построить физическую модель в ERWin?
9. В чем состоят отличия физической модели от логической (в ERWin)?
10. 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?
11. Может ли логическая модель содержать элементы (сущности, атрибуты.), которые не переносятся в физическую модель (в ERWin)? Если да, то чем это может быть вызвано?
12. Что называется «целевой СУБД»? Как можно выбрать целевую СУБД?
13. К каким изменениям в физической модели приведет смена целевой СУБД?
14. В каких нотациях может быть построена физическая модель? Чем они отличаются друг от друга?
15. Как и какие ограничения целостности можно задавать в AllFusion ERwin Data Modeler?
16. Что такое реверс-инжиниринг, как и для каких целей он может использоваться?
17. Как с помощью AllFusion ERwin Data Modeler можно создавать базы данных?

Глава 5.

Создание БД в Microsoft Access 2007

5.1. Общие понятия. Интерфейс

Базой данных в MS Access называется совокупность таблиц, форм, отчетов, запросов, модулей, макросов. Все они являются объектами базы данных. Вся эта совокупность запоминается в одном файле базы данных. В Access 2007 этот файл имеет расширение accdb. Максимальный размер файла базы данных Microsoft Access – 2 Гбайт. Максимальное число объектов в базе – 32 768.

В дальнейшем будут рассмотрены только вопросы создания таблиц и связей между ними, т.е. то, что традиционно понимается под реляционной базой данных.

В более ранних версиях Access для работы с объектами базы данных использовалось окно базы данных. В Access 2007 для этих целей используется новое средство – область переходов. Область переходов также может заменить кнопочные формы – экраны, используемые для переходов по базе данных и выполнения различных задач, например запуска отчетов.

Интерфейс Access 2007 значительно отличается от интерфейса предыдущих версий системы.

При создании новой базы данных после запуска Access надо выбрать кнопку 

(рис. 5.1).

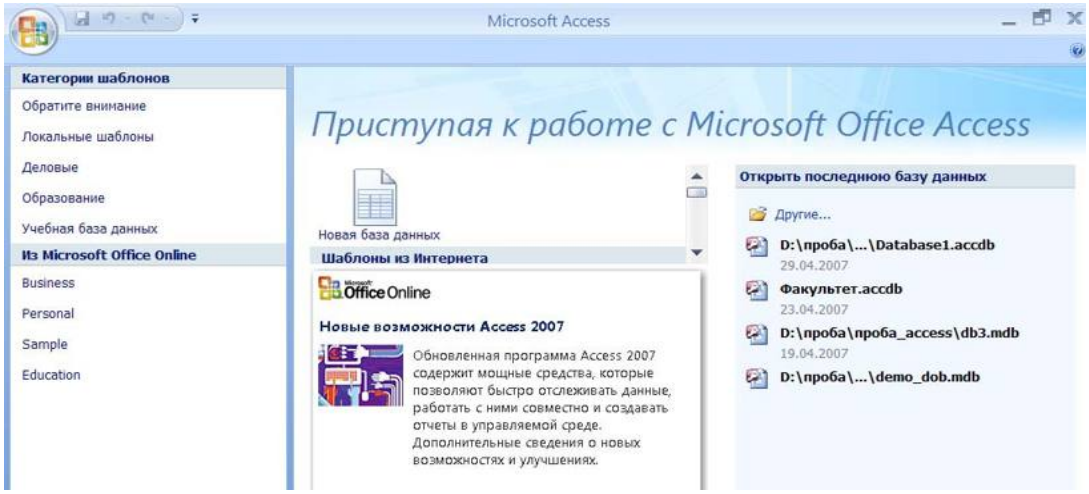


Рис. 5.1. **Начальный вид экрана «Приступая к работе с Microsoft Office Access»**

В результате появится вкладка «***Приступая к работе с Microsoft Office Access***» (рис. 5.2). В правом нижнем углу экрана можно задать имя создаваемого файла БД и определить место, где он будет храниться, после чего нажать кнопку «Создать». Чтобы определить место расположения файла базы данных, следует воспользоваться кнопкой  . В нашем примере для файла базы данных задано имя «Демонстрационная», и он хранится в папке D:\проба\Access_2007.

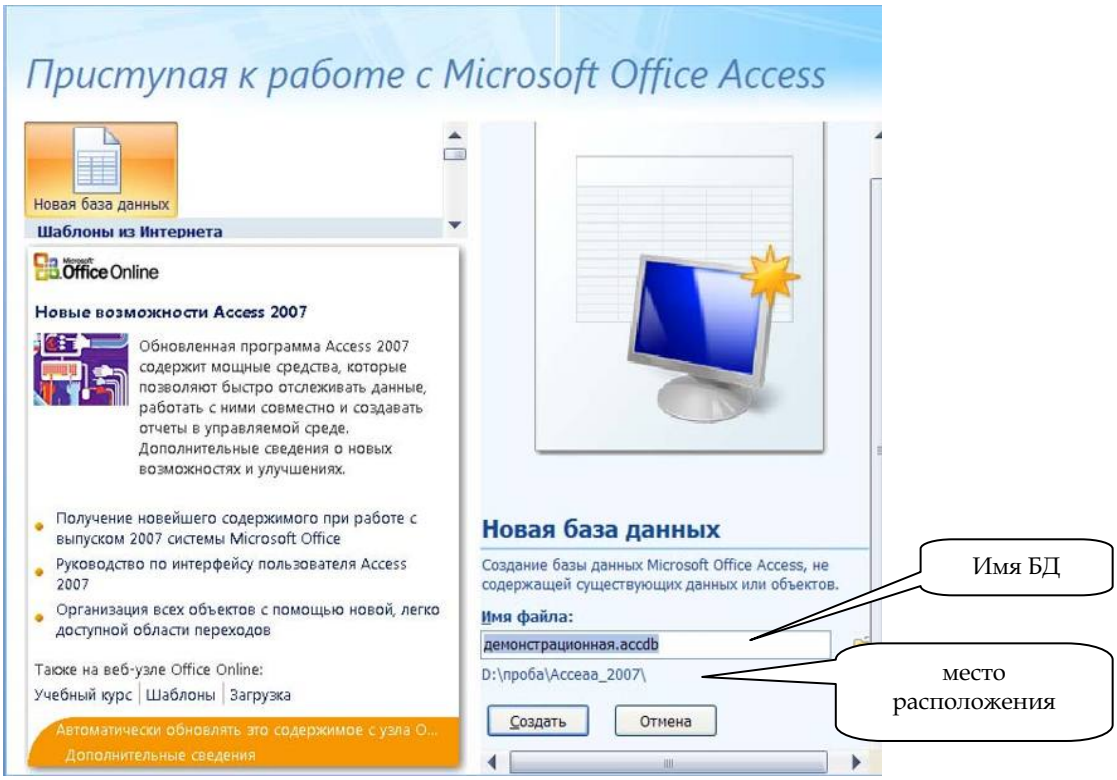


Рис. 5.2. **Создание новой базы данных (задание имени базы данных и места его хранения)**

После выполнения этих шагов появится экран «[название]: база данных», открытый на вкладке ***Режим таблицы*** (рис. 5.3). Это вполне естественно, так как создание базы данных начинается с создания таблиц, в которых и будет храниться информация о предметной области. Существует несколько режимов работы с таблицами и ***Режим таблицы*** является одним из самых простых.

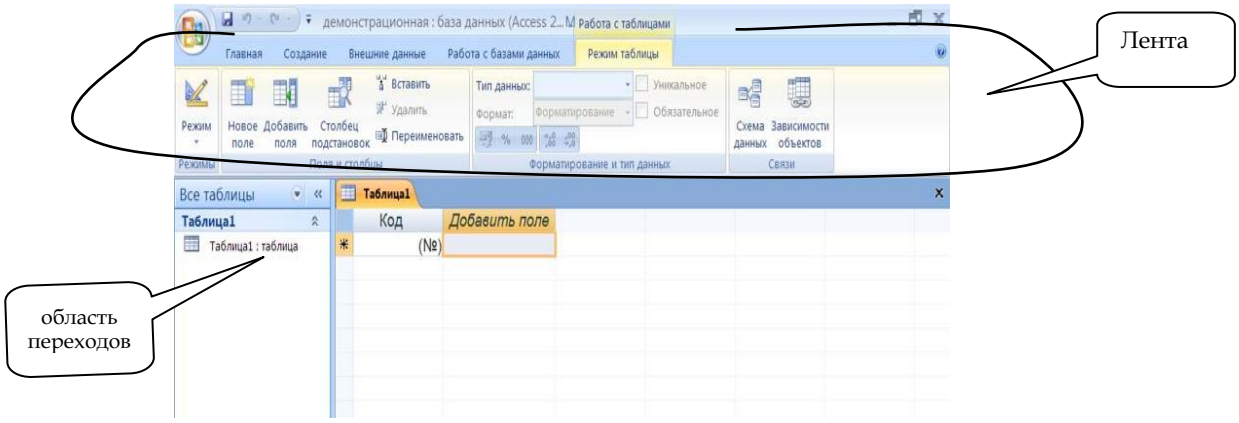


Рис. 5.3. **Создание таблицы в режиме таблицы**

В верхней части появившегося экрана имеется широкая полоса, на которой расположены группы команд. Эта полоса называется ***лента***. На ней находятся все команды, которые в предыдущих версиях Access были размещены в различных меню и панелях

инструментов. Вкладки ленты объединяют логически связанные команды. В Office Access 2007 основными вкладками ленты являются: Главная, Создание, Внешние данные и Работа с базами данных. Каждая вкладка содержит группу связанных команд, которые могут открывать другие новые элементы интерфейса. Лента служит основным командным интерфейсом в Office Access 2007.

Считается, что работа с Access 2007 стала проще, чем наблюдалось в предыдущих версиях системы (Мюррей, 2007), и это достигается и новым интерфейсом, и большим количеством разработанных полнофункциональных шаблонов. Однако и возможность создания таблиц в режиме таблицы, и создание таблиц с помощью мастера таблиц были и в Access 2003. Но в Access 2007 они существенно расширены и стали удобнее.

5.2. Создание таблиц

5.2.1. Общие сведения

Таблицы в Access можно создавать разными способами (будут рассмотрены далее). Для каждой таблицы задается ее имя. Максимальное число знаков в имени таблицы равно 64.

Таблица состоит из полей. Максимальное число полей в таблице – 255. Для каждого поля задается его имя. Максимальное число знаков в имени поля также равно 64. Каждое поле имеет определенный тип данных. В зависимости от способа создания таблицы тип данных может задаваться создателем таблицы путем явного выбора, определяться исходя из типа вводимых данных или определяться типом данных, служащих источником данных при создании данной таблицы.

Согласно реляционной теории каждая таблица должна иметь ключ[1]. Access разрешает отказаться от определения первичного ключа в таблице. Ключ может быть задан проектировщиком в явном виде, а может быть создан системой автоматически. Ключ может быть как простым, состоящим из одного поля, так и составным.

По полям, объявленным ключом, система автоматически создает индекс. Индексы могут быть построены и для вероятных ключей (уникальный индекс), так и по полям, не являющимся уникальными. Максимальное число индексов в таблице – 32. Максимальное число полей в индексе – 10.

Каждое поле имеет набор свойств. Состав этого набора зависит от данных этого поля.

5.2.2. Создание таблицы в режиме таблицы

В режим создания таблицы в режиме таблицы можно попасть и при выборе позиции  Таблица на вкладке Создание (рис. 5.4).

[1] Понятия «ключ», «первичный ключ», «альтернативный ключ», «внешний ключ» должны быть известны из ранее изученных тем курса, так же как и вопросы проектирования реляционных баз данных. В данном разделе рассматриваются только вопросы создания уже спроектированных баз данных в среде СУБД Access.

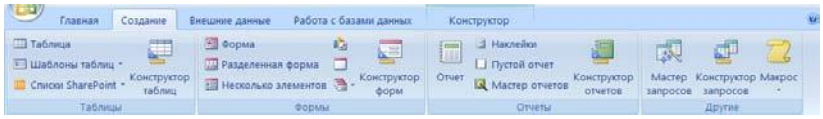


Рис. 5.4. **Вкладка «СОЗДАНИЕ»**

При создании таблицы в режиме таблицы можно и задавать имена полей, и определять тип данных, а также указать признак уникальности и обязательности поля. Тип данных можно выбрать из ниспадающего списка (рис. 5.5).

Имя поля задается в верхней строке высвеченной таблицы. Во вторую строку можно вводить значение поля. Автоматически введенное в таблицу поле Код имеет тип Счетчик и объявляется ключом таблицы.

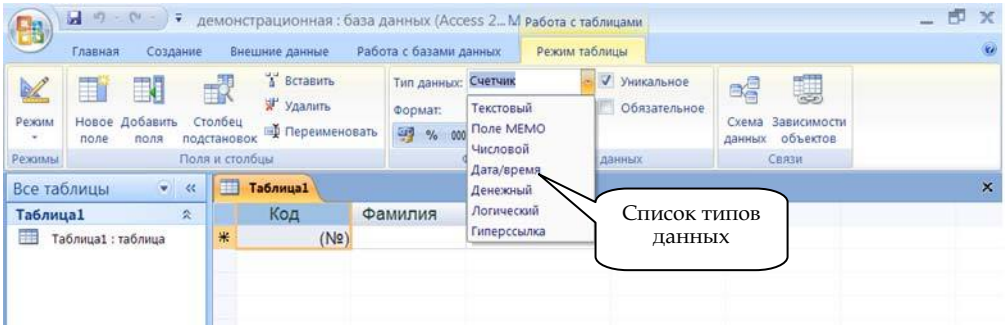


Рис. 5.5. **Создание таблицы в режиме таблицы. Выбор типа поля**

Режим таблицы используется не только для описания структуры таблицы, но и для ввода в нее данных, также просмотра данных. Более того, создание таблицы можно начать не с описания ее структуры, а с заполнения ее данными. Следует отметить, что тип поля может выбираться не только из ниспадающего списка, но и определяться системой автоматически, в соответствии с типом введенного значения.

Интересно заметить, что с полем типа Дата ассоциирован Календарь (рис. 5.6), который может быть использован как для ввода, так и для просмотра данных. Однако недостатком календаря является то, что он позволяет быстро менять месяц, но не позволяет менять год при выборе нужной даты.

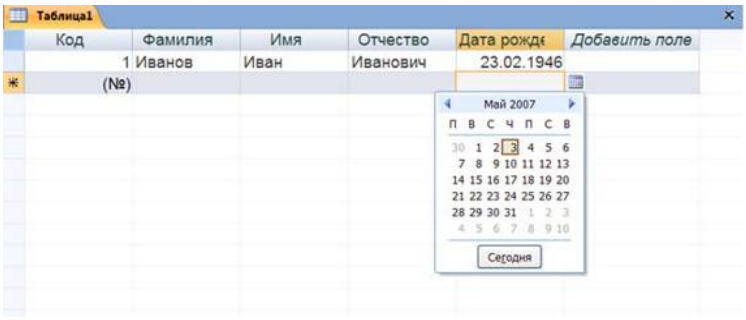


Рис. 5.6. **Календарь, ассоциированный с полем типа Дата**

Спроектированная структура таблицы может быть изменена. Для выбора необходимого действия можно воспользоваться контекстным меню (рис. 5.7).

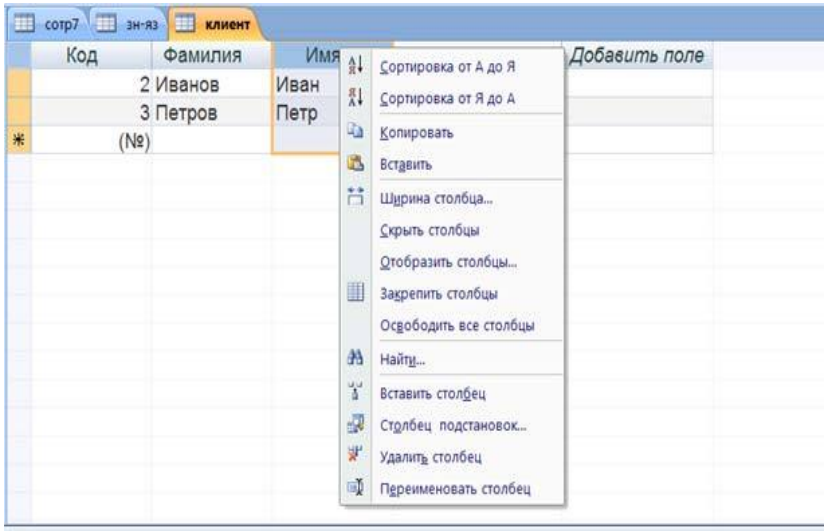


Рис. 5.7. **Контекстное меню при работе с таблицей в режиме таблицы**

Таблица может содержать большое количество полей, что затрудняет просмотр записей. Для того чтобы сделать просмотр более удобным, можно выделить столбцы, которые будут высвечиваться в начале таблицы и не перемещаться при продвижении от поля к полю, и далее выбрать позицию меню **«Закрепить столбцы»**.

На первый взгляд создание таблицы в режиме таблицы является простым, наглядным и наиболее естественным способом создания таблиц. Но при этом могут возникнуть проблемы, которые трудно объяснить, не вникая в особенности проектирования баз данных.

Режим таблицы не является единственным возможным способом создания таблицы.

Рис. 5.8 на вкладке **Создание** представляет несколько способов создания таблиц. Наиболее универсальным и представляющим наибольшие возможности является режим **Конструктора таблиц**.

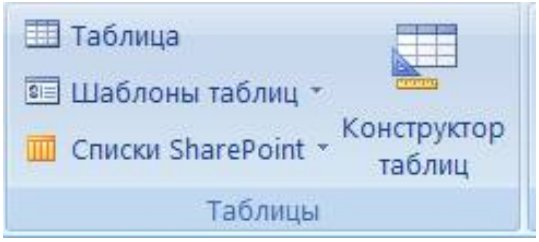


Рис. 5.8. **Блок Таблицы**

5.2.3. Создание таблицы в режиме Конструктора

5.2.3.1. Общая характеристика. Типы полей

При выборе режима «**Конструктор таблиц»** появится вкладка для описания структуры таблицы и других ее характеристик (рис. 5.9).

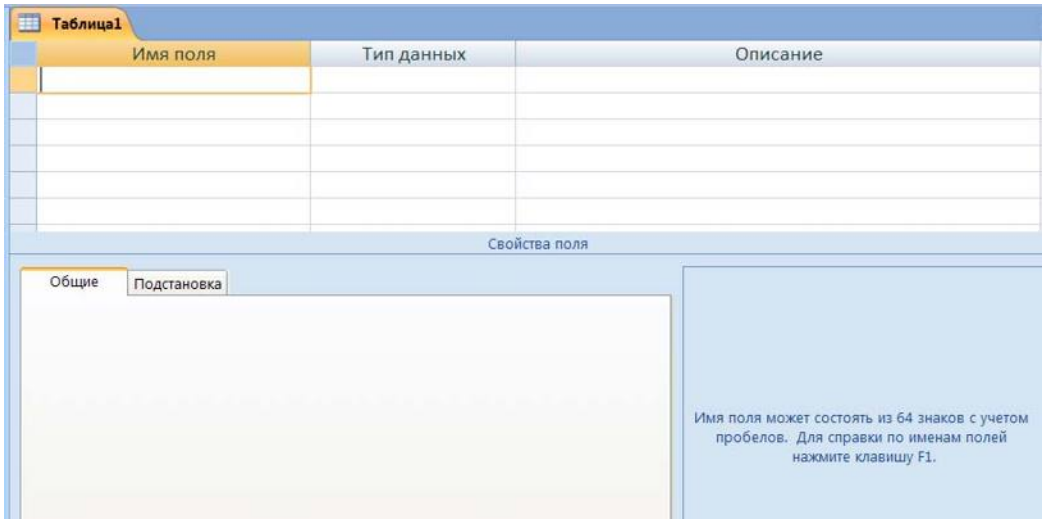


Рис. 5.9. **Вид экрана при описании таблицы в режиме Конструктора**

На этой вкладке в табличной форме надо 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[1]. Вообще к заданию длинных имен на русском языке надо относиться с осторожностью, особенно, если есть вероятность, что создаваемое приложение будет в дальнейшем использоваться в распределенных гетерогенных системах.

При задании имен не допускайте их совпадения 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, избегайте этого 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 поле с таким же названием есть в таблице «КАФЕДРА», то в первом случае это будет код сотрудника, а во втором – код кафедры. Многие системы (и Access в том числе) автоматически связывают таблицы по полям, которые имеют одинаковые имя, тип и длину. Если имена даны непродуманно, то могут либо возникнуть неправильные связи, либо процесс задания связей будет несколько сложнее, чем при правильном задании имен.

После задания имени надо выбрать тип поля. Если щелкнуть мышкой по свободной ячейке графы **«Тип поля»**, то высветится список допустимых типов полей (рис. 5.10), из которого и следует выбрать подходящий для описываемого поля тип.

[1] В данном учебном пособии длинные имена с пробелами даются исключительно с целью достижения большей наглядности излагаемого материала.

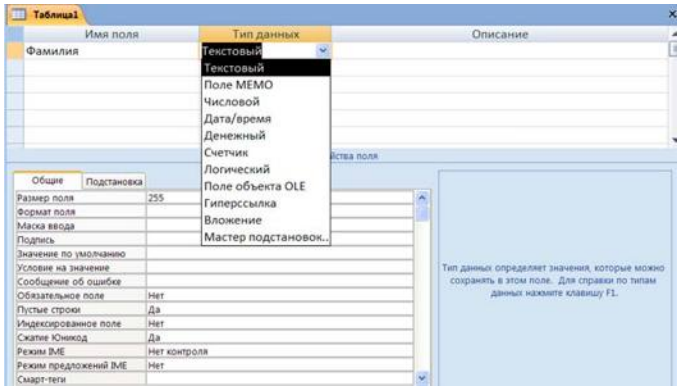


Рис. 5.10. **Типы данных**

Допустимые типы данных в Access 2007 и их краткая характеристика приведены в табл. 5.1.

Таблица 5.1

Типы данных в Access 2007

Тип данных	Содержимое поля	Размер
Текстовый	Текст или числа, не требующие проведения расчетов, например, номера телефонов, коды и т.п.	Максимальное число символов – 255
Поле MEMO	Длинный текст или комбинация текста и чисел.	До 65535 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 байт
Логический	Поля, которые могут содержать одно из двух возможных значений (True/False, Да/Нет).	1 бит
Поле объекта OLE	Объект, связанный или внедренный в таблицу Microsoft Access.	До 1 Гбайт (ограничивается объемом диска).
Гиперссылка	Строка, состоящая из букв и цифр, и представляющая адрес гиперссылки. Адрес гиперссылки может состоять максимум из трех частей: *текст* – текст, выводимый в поле или в элементе управления; *адрес* – путь к файлу (в формате пути UNC) или странице (адрес URL); *дополнительный адрес* – смещение внутри файла или страницы.	Каждая из трех частей в типе Гиперссылка может содержать до 2048 символов.
Вложения	Файлы произвольного типа	

Новым по сравнению с Access 2003 является тип данных **Вложения**. Функция вложения в Access 2007 служит для добавления одного или нескольких файлов произвольного типа в записи базы данных. Например, в таблице СОТРУДНИК можно добавить к записи каждого сотрудника резюме, фотографию, тексты приказов, относящихся к нему.

В более ранних версиях Access для хранения изображений и документов использовалась технология OLE (Object Linking and Embedding — связывание и внедрение объектов). Эта возможность сохранена и в Access 2007. По умолчанию с помощью технологии OLE создается растровый эквивалент изображения или документа. Такие растровые файлы могут быть слишком большими — иногда в несколько раз больше исходного файла. При просмотре изображения или документа из базы данных с помощью технологии OLE отображалось растровое изображение, а не исходный файл. Вложения же позволяют хранить данные более рационально. При использовании вложений файлы, не являющиеся изображениями, открываются в соответствующих программах, так что эти файлы можно находить и редактировать непосредственно в приложении Access.

В списке допустимых типов полей (см. рис. 5.10) 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, будет 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.

Имя и тип поля должны задаваться обязательно. Графа **«Описание»** может не заполняться. Эта графа используется в целях документирования проекта.

Предположим, что мы создаем таблицу, содержащую сведения о профессорско-преподавательском составе вуза. Состав и тип полей создаваемой таблицы представлены на рис. 5.11.

Имя поля	Тип данных
ФИО	Текстовый
Дата_рождения	Дата/время
Пол	Текстовый
Код_кафедры	Числовой
Дата_приема_на_работу	Дата/время
Оклад	Денежный
В_о	Логический
Должность	Текстовый
▶ Руководитель	Числовой

Рис. 5.11. **Состав полей таблицы «СОТРУДНИК»**

5.2.3.2. Использование мастера подстановки

Обратим внимание на поле «Должность». Для выбранной категории сотрудников имеется всего пять возможных должностей: ассистент, старший преподаватель, доцент, профессор и заведующий кафедрой. 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. Начиная с Access'7 стало возможным задать его и при описании таблицы.

Используем «**Мастер подстановок**» при определении типа данных поля «Должность». Для этого можно либо при выборе типа указать «**Мастер подстановок**» (см. последнюю строку в ниспадающем списке типов полей на рис. 5.10), либо использовать кнопку – «Столбец подстановки». «Последовательность шагов при создании поля подстановки изображены на рис. 5.12–5.15. При создании поля с помощью мастера подстановок имя поля можно не задавать, а сразу перейти к столбцу **«Тип данных»** и выбрать в списке строку «**Мастер подстановок»**. Имя поля будет задано позже в процессе создания поля с помощью мастера.

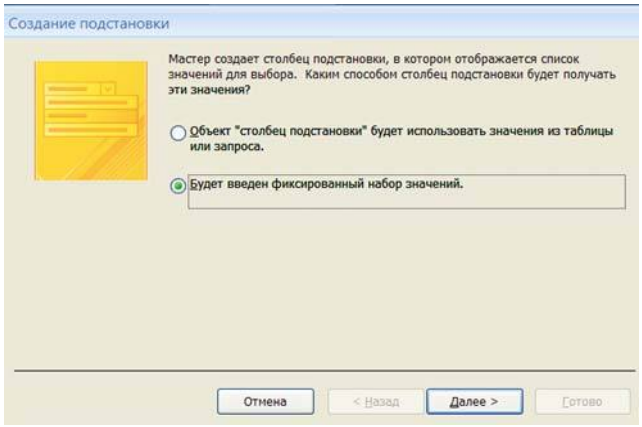


Рис. 5.12. **Создание столбца подстановки. Начальный экран**

Так как список, создаваемый в рассматриваемом случае, короткий и стабильный, то выберем возможность ввода фиксированного набора значений (рис. 5.12). В появившемся далее окне введем требуемые значения (рис. 5.13).

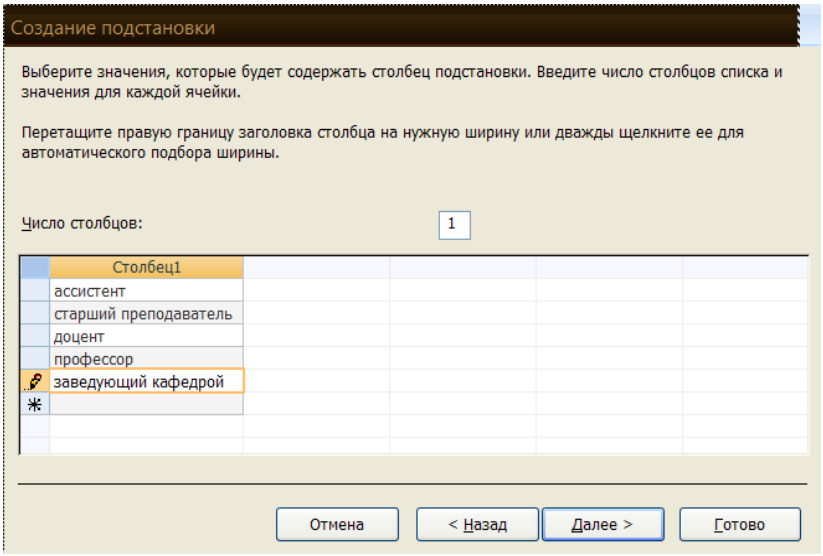


Рис. 5.13. **Создание столбца подстановки. Столбец с введенным списком значений**

Далее зададим имя этого поля (рис. 5.14).

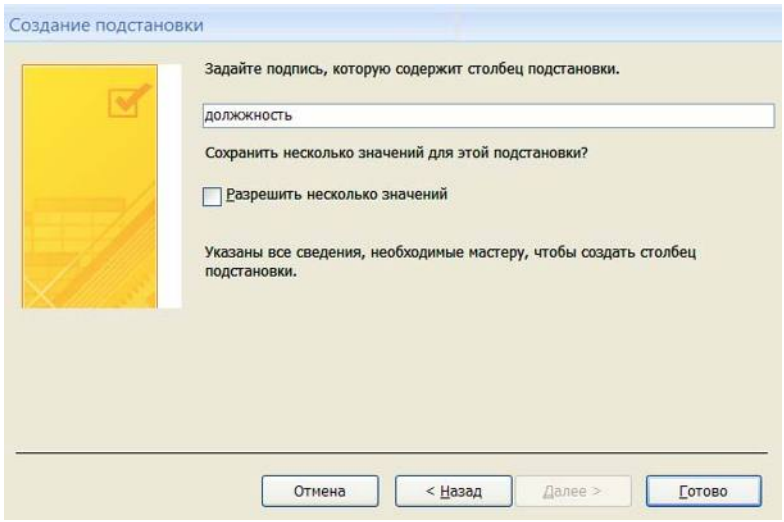


Рис. 5.14. **Создание столбца подстановки. Задание имени столбца**

При создании поля таким способом его тип будет «текстовый», а длина его будет соответствовать установленной длине текстового поля по умолчанию. После создания поля с использованием мастера подстановок с фиксированным набором значений его тип и длину можно скорректировать.

По сравнению с предшествующими версиями Access появилась новая возможность «Разрешить несколько значений».

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**» (рис. 5.15) 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.

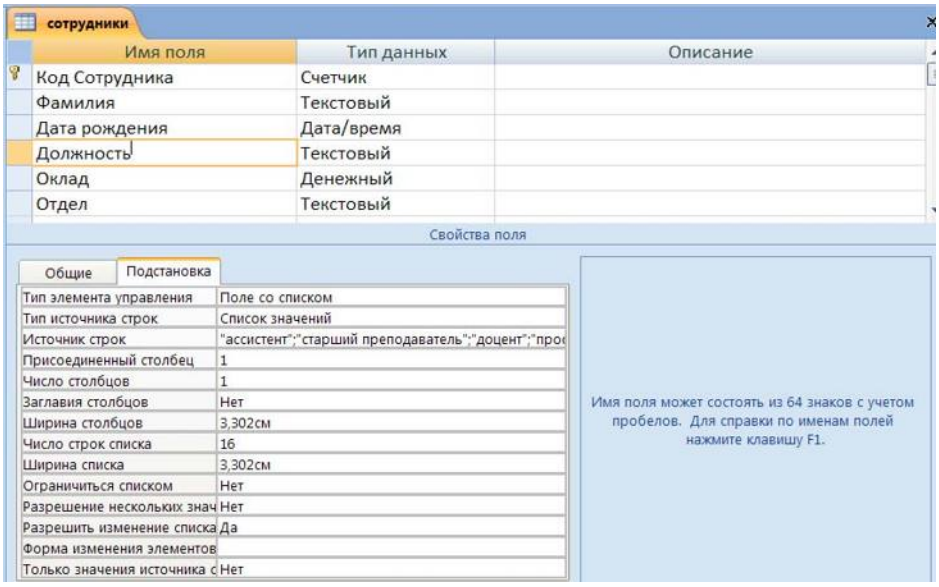


Рис. 5.15. **Свойства подстановки**

Если вдруг оказалось, что список подстановки надо скорректировать после его создания, то можно нажать кнопку  в строке «Источник строк» и в появившемся окне «Изменение элементов списков» внести требуемые изменения. В этом же окне можно указать «Значение по умолчанию» (рис. 5.16).

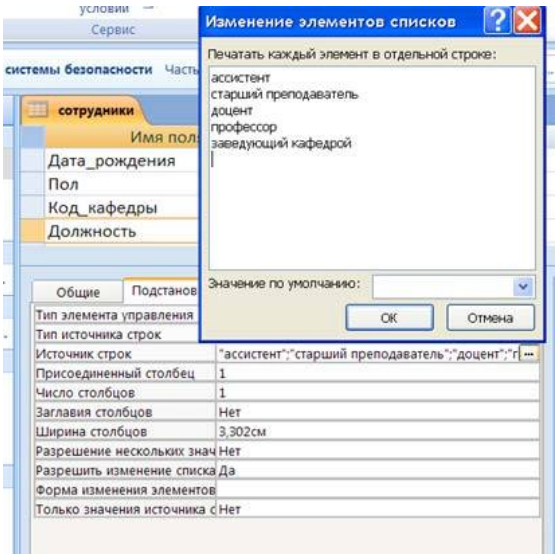


Рис. 5.16. **Изменение списка подстановки**

Использование списка подстановки имеет смысл практически только для полей типа «Текстовый». Для полей всех других типов, кроме «Логический», система выдает предупреждающее сообщение (рис. 5.17).

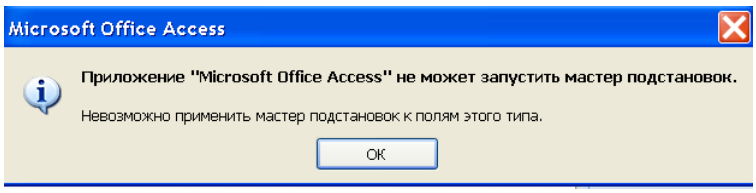


Рис. 5.17. **Сообщение о недопустимости действия**

Если число значений поля подстановки достаточно велико и они могут меняться со временем, то следует использовать другую альтернативу – «**Столбец подстановки использует значения из таблицы или запроса**» (см. рис. 5.12). Эта возможность используется в нашем примере для поля КОД_КАФЕДРЫ, значения которого будут браться из таблицы КАФЕДРА. Естественно, что таблица КАФЕДРА должна быть предварительно создана[1].

Для создания поля подстановки, источником для которого служит другая таблица, лучше сначала создавать основную таблицу (в паре «КАФЕДРА – СОТРУДНИК» основной будет таблица КАФЕДРА), а затем – подчиненную (в нашем примере –

[1] Если вы изучаете данное пособие, 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. После этого откройте таблицу СОТРУДНИК в режиме «конструктор» и выполните описанные ниже шаги. При этом система автоматически свяжет указанные таблицы.

СОТРУДНИК). Если же сначала, как в нашем случае, создавалась подчиненная таблица, то описание поля, для которого источником подстановки будет другая таблица, может быть впоследствии скорректировано.

При создании подстановки из другой таблицы/запроса надо выбрать таблицу/запрос, которая будет являться источником данных для описываемого поля (рис. 5.18).

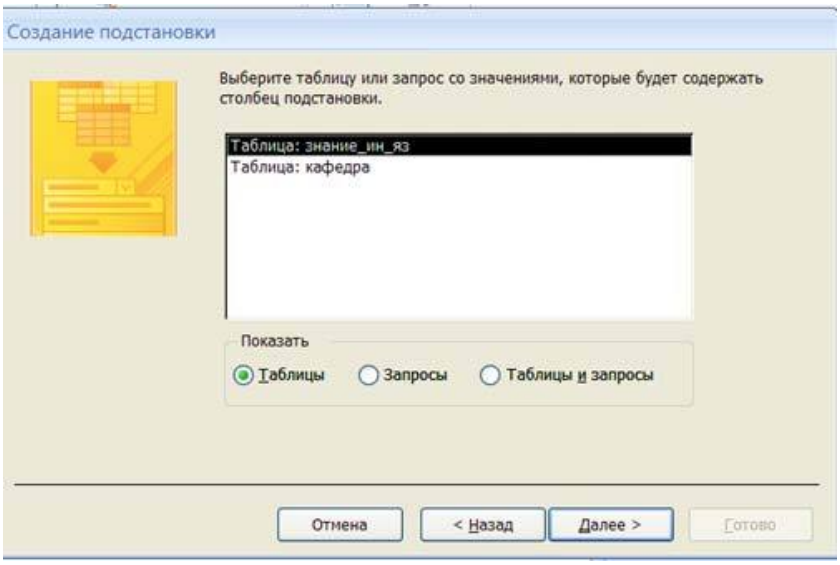


Рис. 5.18. **Выбор таблицы/запроса источника для поля подстановки**

Далее надо определить колонку таблицы-источника, значения из которой будут подставляться в описываемую колонку (рис. 5.19). 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.

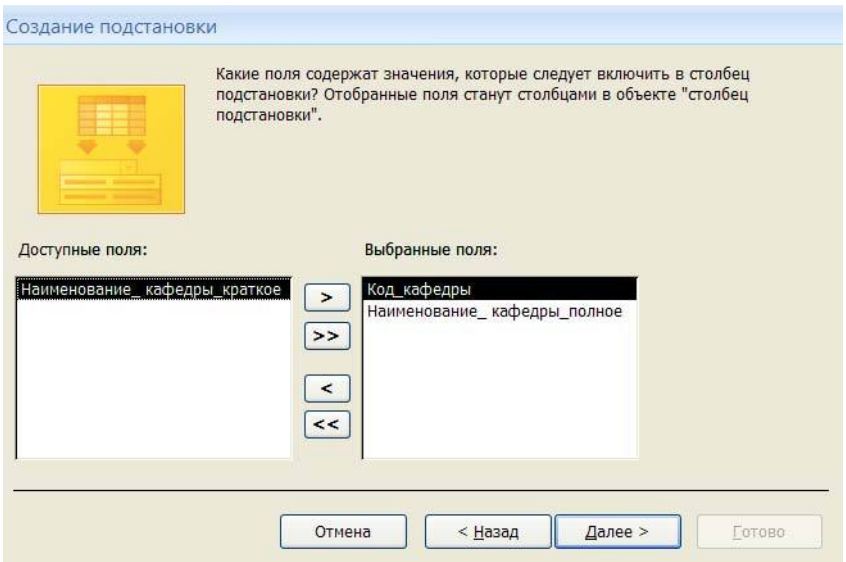


Рис. 5.19. **Выбор колонки-источника для поля подстановки**

На следующем шаге (рис. 5.20) возможно задать признак упорядочения значений для поля подстановки.

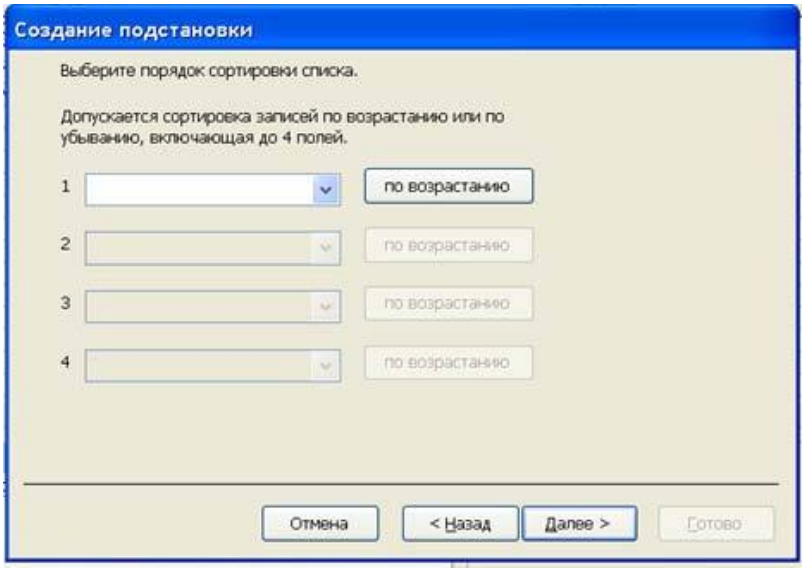


Рис. 5.20. **Выбор порядка сортировки**

В нашем случае удобно упорядочить по полю НАИМЕНОВАНИЕ_КАФЕДРЫ_ ПОЛНОЕ (рис. 5.21).

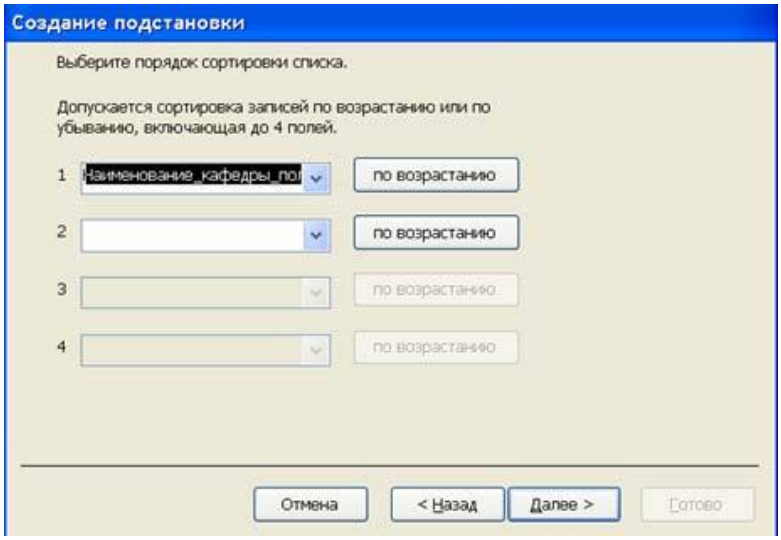


Рис. 5.21. **Выбранное значение поля сортировки**

В появившемся далее окне (рис. 5.22) 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 НАИМЕНОВАНИЕ_КАФЕДРЫ_ПОЛНОЕ. Если эту «галочку» убрать, то будут выводиться оба поля: КОД_КАФЕДРЫ и НАИМЕНОВАНИЕ_ КАФЕДРЫ_ПОЛНОЕ.

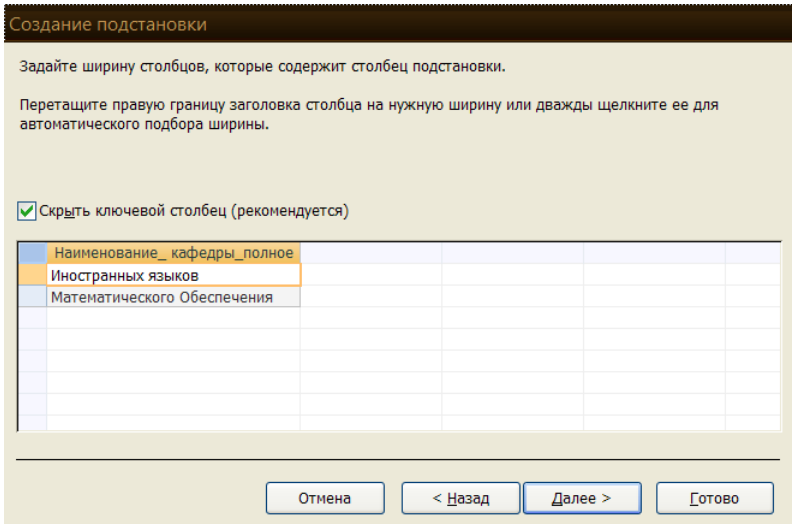


Рис. 5.22. **Создание поля подстановки. Задание ширины столбцов**

При таком способе создания таблицы, несмотря на то, что реально в таблице будет храниться именно код кафедры, при просмотре таблицы в режиме таблицы будет выводиться не код кафедры, а ее наименование.

5.2.3.3. 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 при описании таблицы ключ не был задан, то при переходе в режим таблицы или закрытии файла появится соответствующее предупреждение (рис. 5.23).

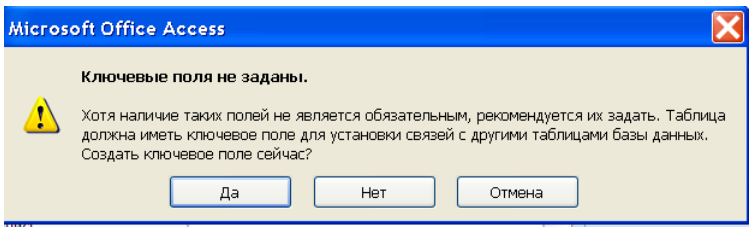


Рис. 5.23. **Задание ключевого поля таблицы при закрытии таблицы**

Если вы собираетесь 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. Access создаст это поле автоматически.

Если вы определяете ключ самостоятельно, то это можно сделать несколькими путями: позиционироваться на соответствующее поле и нажать кнопу «**Ключевое поле**» (

) на ленте вкладки Конструктор (рис. 5.24) либо воспользоваться правой кнопкой мыши для вызова контекстного меню, предварительно позиционировавшись на то поле, которое определяется как ключевое.

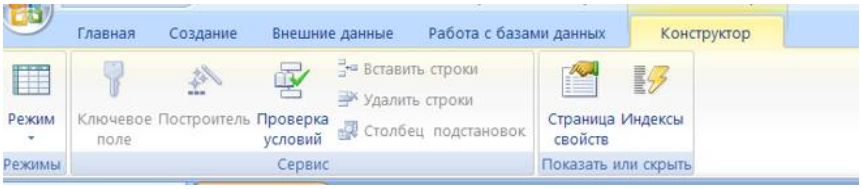


Рис. 5.24. **Лента вкладки Конструктор**

Как известно, ключ может быть составным. Чтобы определить составной ключ, надо выделить соответствующую совокупность полей, а затем выполнить те же действия, что и при задании простого ключа.

5.2.3.4. Свойства полей

В нижней части экрана описания таблицы отображается список свойств выбранного поля. Перечень свойств будет зависеть от выбранного типа поля (табл. 5.2).

Таблица 5.2

Свойства полей (в зависимости от типа поля)

Тип поля / Свойство	Текстовое	Логическое	MEMO	Числовое	Дата/время	Денежный	Счетчик
Размер поля	+			+			+
Число десятичных знаков				+		+	
Формат поля	+	+		+	+		+
Формат текста			+			+	
Маска ввода	+			+		+	
Подпись	+	+	+	+	+	+	+
Значение по умолчанию	+	+	+	+	+	+	
Условие на значение	+	+	+	+	+	+	
Сообщение об ошибке	+	+	+	+	+	+	
Обязательное поле	+	+	+	+		+	
Пустые строки	+		+				
Индексированное поле	+	+	+	+	+	+	+
Сжатие Юникод	+		+				
Режим IME	+		+				
Режим предложений IME	+		+				
Смарт-теги	+		+	+		+	+
Выравнивание текста	+	+	+	+	+	+	+

Окончание табл. 5.2

Тип поля / Свойство	Текстовое	Логическое	МЕМО	Числовое	Дата/время	Денежный	Счетчик
Только добавление			+				
Новые значения							+
Отображать элементы выбора дат					+		

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.

Некоторые из свойств полей понятны без дополнительных пояснений. Некоторые мы поясним ниже на примерах.

Свойство «**Индексированное поле**» определяет, надо ли создавать индекс по этому полю. Индекс ускоряет выполнение запросов, в которых используются индексированные поля, и операции сортировки и группировки.

Свойство **«Индексированное поле»** может иметь следующие значения (табл. 5.3):

Таблица 5.3

Значения	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 нами примере в связи с тем, что по полю ФАМИЛИЯ часто осуществляется поиск и упорядочение информации, желательно по нему произвести индексацию. Так как среди сотрудников возможны однофамильцы, то должны быть разрешены совпадения значений индексируемого поля.

Не допускается создание индексов для полей МЕМО, гиперссылок и объектов OLE.

Ключевое поле КОД_СОТРУДНИКА имеет тип «**Счетчик**». Только для полей этого типа имеется свойство «**Новые значения**». Оно определяет способ определения 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 Длинное целое.

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). 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).

Совет: 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 пустые строки.

5.2.3.5. Сохранение описания таблицы

После того как описание таблицы завершено, его надо сохранить. Этого можно достигнуть разными путями: выбрать в контекстном меню (рис. 5.25) позицию «**Сохранить»** и в появившемся окне ввести имя таблицы или позицию «**Закрыть»** (после чего на вопрос «**Сохранить изменения макета или структуры?**» ответить «**Да**»), или перейти в «**Режим таблицы**» и на сообщение «**Сначала необходимо сохранить таблицу. Сделать это сейчас?**» ответить «**Да**» (этот способ надо использовать когда вы хотите сразу после описания структуры таблицы вводить данные в эту таблицу). В появившемся после указанных действий окошке следует ввести имя созданной таблицы.

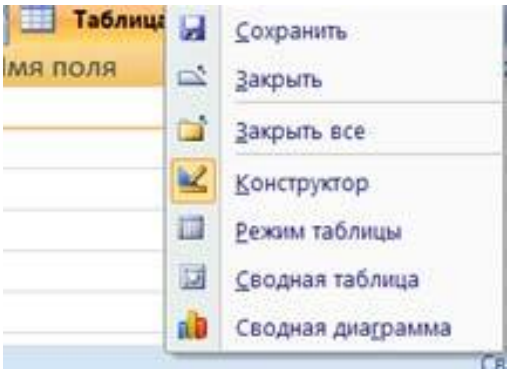


Рис. 5.25. **Контекстное меню 2**

Кроме описания структуры таблицы в окне **«Свойства»** таблицы (рис. 5.26) есть возможность дать ее неформализованное описание. Такая возможность способствует лучшему документированию и, как следствие, – лучшему пониманию системы.

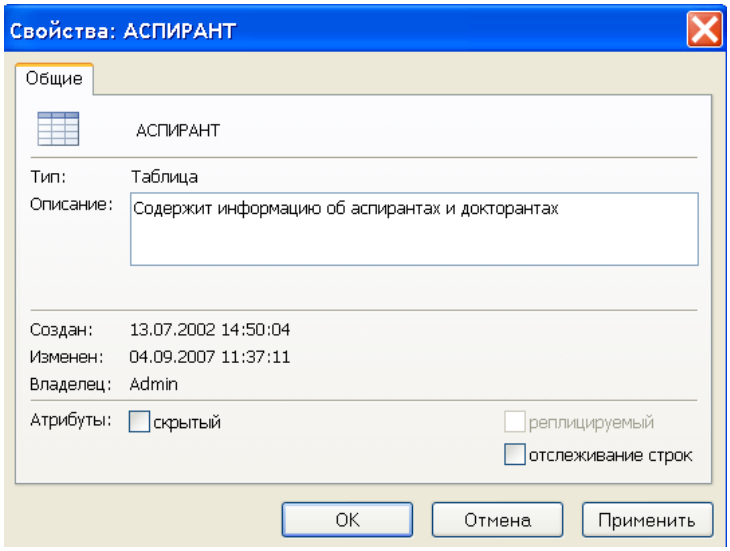


Рис. 5.26. **Окно «Свойства» объекта базы данных**

5.2.3.6. Создание таблиц для контрольного примера

Аналогичные действия повторяются при создании остальных таблиц БД.

Создадим таблицы КАФЕДРА со структурой, представленной на рис. 5.27, и ДЕТИ со структурой, представленной на рис. 5.28.

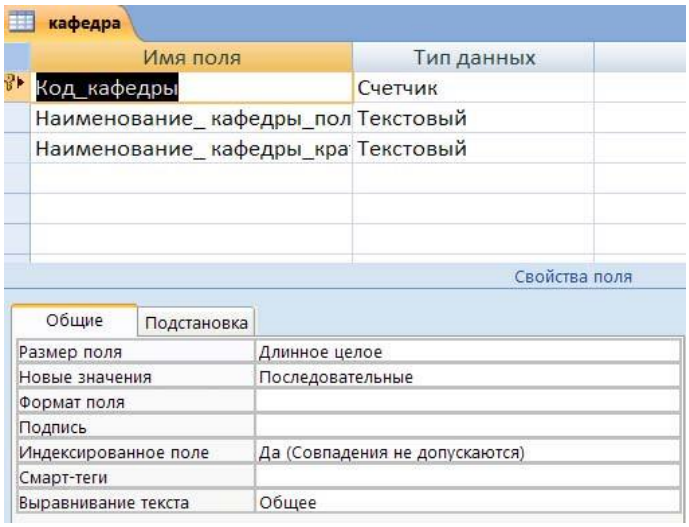


Рис. 5.27. **Структура таблицы КАФЕДРА**

При создании таблицы КАФЕДРА ключ «Код_Ккафедры», как и в случае с таблицей СОТРУДНИК, создадим автоматически при закрытии таблицы.

Рис. 5.28. **Структура таблицы ДЕТИ**

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 одноименном поле 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 «числовое» и размер поля – «длинное целое».

Созданные таблицы будут использоваться далее для иллюстрации возможностей создания запросов и отчетов.

5.2.3.7. 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 внести изменения прямо в режиме **Таблица** либо перейти в режим «**Конструктора»**, воспользовавшись кнопкой «**Режим»**. В режиме **Таблица** можно произвести не все манипуляции со структурой таблицы; режим **Конструктора** является более мощным и универсальным.

Если нужная таблица уже 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 в контекстном меню[1] выберите строку «**Добавить строки**», 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 до изменения типов данных и размеров полей рекомендуется сделать ее копию, так как несовместимость существующих данных с новым значением свойства «**Тип данных**» может привести к потере данных.

[1] 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.

5.2.4. Другие способы создания таблиц

5.2.4.1. Копирование структуры таблицы

Если вы создаете таблицу, структура которой имеет много общего со структурой ранее созданной таблицы, то можно скопировать структуру существующей таблицы (для этого надо позиционироваться на соответствующей таблице, выбрать позицию контекстного меню «**Копировать**», потом – «**Вставить**», после чего в появившемся окне (рис. 5.29) 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.

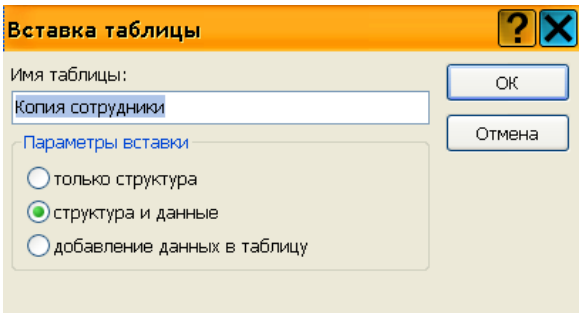


Рис. 5.29. **Копирование таблицы**

5.2.4.2. Создание таблиц на основе шаблона

Кроме того, создать таблицу можно на основе шаблона таблиц, выбрав кнопку **Шаблон таблиц** блока **Таблицы** на вкладке **Создание** (см. рис. 5.8). В Office Access 2007 включены шаблоны таблиц Контакты, Задачи, Вопросы, События, Основные фонды. При выборе какого-то из шаблонов соответствующая таблица открывается в режиме таблицы.

Далее можно откорректировать структуру в обычном порядке. Как мы видим, использование этой возможности 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.

Другая возможность использования шаблонов заключается в том, чтобы создать по шаблону не отдельную таблицу, а базу данных. Для этого при запуске Access (см. рис. 5.1) можно позиционироваться на строке «Локальные шаблоны» и далее выбрать подходящий шаблон (рис. 5.30). При этом будут созданы не только таблицы, но и связанные с ними таблицы, экранные формы и, возможно, другие объекты.

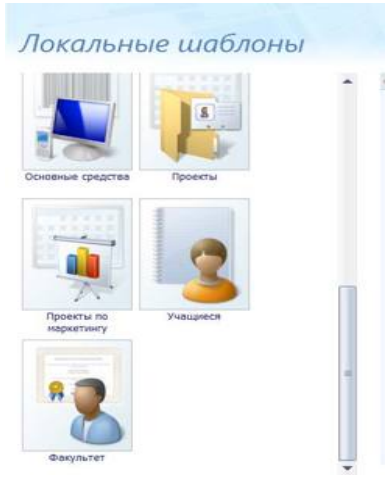


Рис. 5.30. **Шаблоны баз данных**

5.2.4.3. Создание таблицы путем импорта из других систем

Создать таблицу можно и путем импорта ее из других систем. Для этого надо на вкладке **Внешние данные** в группе **Импорт** (рис. 5.31) выбрать один из доступных типов источников данных.

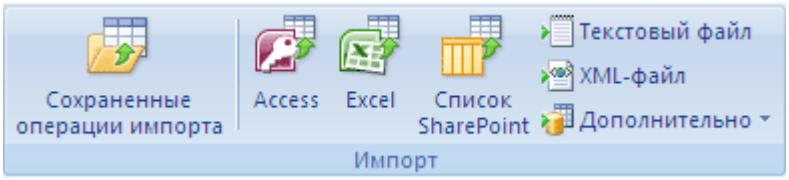


Рис. 5.31. **Импорт данных**

Далее надо выбрать конкретный файл, являющийся источником данных, и указать, где и как данные источника будут храниться в текущей базе данных (рис. 5.32).

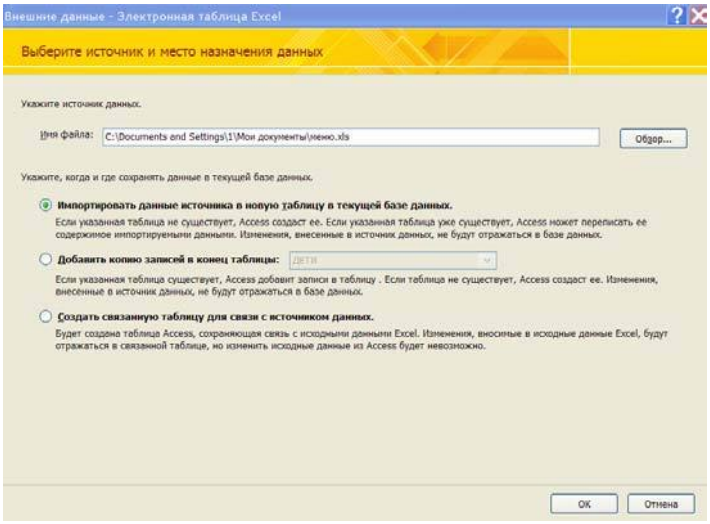
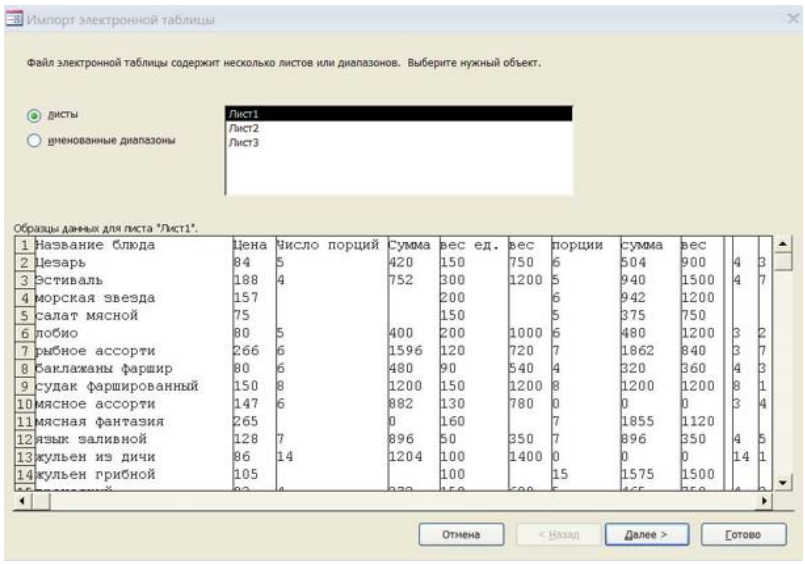


Рис. 5.32. **Выбор источника и места назначения данных**

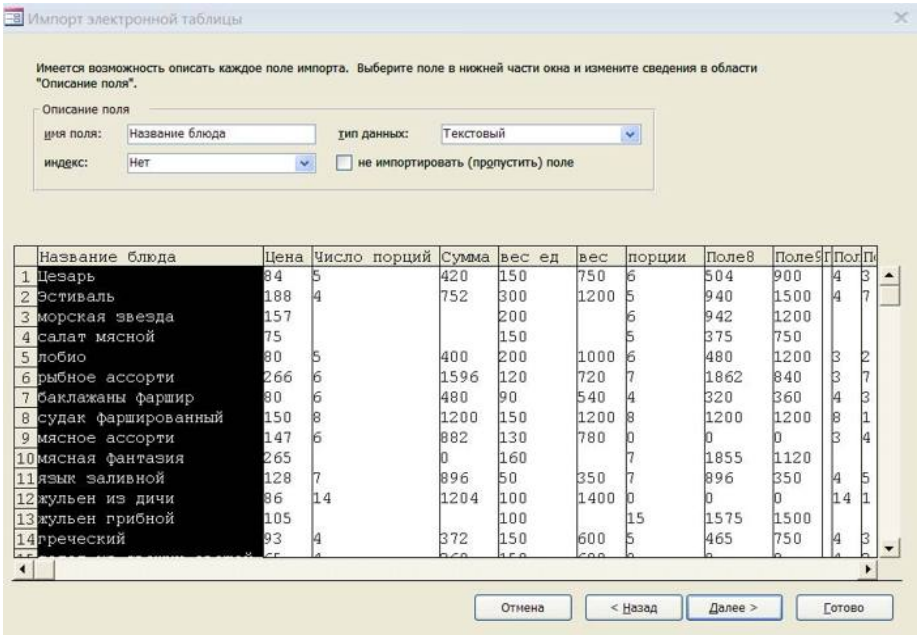
В рассматриваемом примере в качестве источника данных выбрана электронная таблица Excel. На следующем шаге выбирается лист используемой электронной таблицы (рис. 5.33).



	Название блюда	Цена	Число порций	Сумма	вес ед.	вес	порции	сумма	вес		
1	Название блюда	Цена	Число порций	Сумма	вес ед.	вес	порции	сумма	вес		
2	Цезарь	84	5	420	150	750	6	504	900	4	3
3	Эстиваль	188	4	752	300	1200	5	940	1500	4	7
4	морская звезда	157			200		6	942	1200		
5	салат мясной	75			150		5	375	750		
6	лобио	80	5	400	200	1000	6	480	1200	3	2
7	рыбное ассорти	266	6	1596	120	720	7	1862	840	3	7
8	баклажаны фаршир	80	6	480	90	540	4	320	360	4	3
9	судак фаршированный	150	8	1200	150	1200	8	1200	1200	8	1
10	мясное ассорти	147	6	882	130	780	0	0	0	3	4
11	мясная фантазия	265		0	160		7	1855	1120		
12	язык заливной	128	7	896	50	350	7	896	350	4	5
13	жульен из дичи	86	14	1204	100	1400	0	0	0	14	1
14	жульен грибной	105			100		15	1575	1500		

Рис. 5.33. **Импорт электронной таблицы. Выбор листа таблицы**

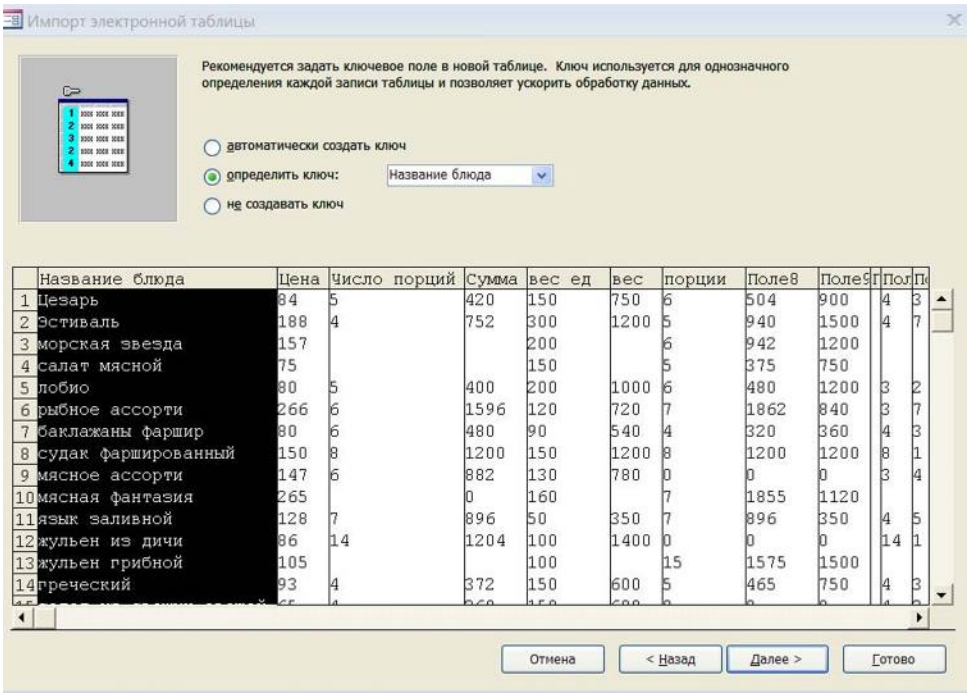
После этого можно воспользоваться возможностью уточнить имена полей таблицы. Если исходная таблица содержала заголовки столбцов, то они станут именами полей. В процессе описания вновь создаваемой таблицы можно изменить имя поля, тип данных, задать индекс. В результатную таблицу могут переноситься не все поля исходной таблицы (рис. 5.34).



	Название блюда	Цена	Число порций	Сумма	вес ед	вес	порции	Поле8	Поле9		
1	Цезарь	84	5	420	150	750	6	504	900	4	3
2	Эстиваль	188	4	752	300	1200	5	940	1500	4	7
3	морская звезда	157			200		6	942	1200		
4	салат мясной	75			150		5	375	750		
5	лобио	80	5	400	200	1000	6	480	1200	3	2
6	рыбное ассорти	266	6	1596	120	720	7	1862	840	3	7
7	баклажаны фаршир	80	6	480	90	540	4	320	360	4	3
8	судак фаршированный	150	8	1200	150	1200	8	1200	1200	8	1
9	мясное ассорти	147	6	882	130	780	0	0	0	3	4
10	мясная фантазия	265		0	160		7	1855	1120		
11	язык заливной	128	7	896	50	350	7	896	350	4	5
12	жульен из дичи	86	14	1204	100	1400	0	0	0	14	1
13	жульен грибной	105			100		15	1575	1500		
14	преческий	93	4	372	150	600	5	465	750	4	3

Рис. 5.34. **Импорт электронной таблицы.**
Уточнение состава результирующей таблицы

На следующем шаге можно определить ключ таблицы (рис. 5.35).



	Название блюда	Цена	Число порций	Сумма	вес ед	вес	порции	Поле8	Поле9	Пол	Пк
1	Цезарь	84	5	420	150	750	6	504	900	4	3
2	Эстиваль	188	4	752	300	1200	5	940	1500	4	7
3	морская звезда	157			200		6	942	1200		
4	салат мясной	75			150		5	375	750		
5	лобио	80	5	400	200	1000	6	480	1200	3	2
6	рыбное ассорти	266	6	1596	120	720	7	1862	840	3	7
7	баклажаны фаршир	80	6	480	90	540	4	320	360	4	3
8	судак фаршированный	150	8	1200	150	1200	8	1200	1200	8	1
9	мясное ассорти	147	6	882	130	780	0	0	0	3	4
10	мясная фантазия	265		0	160		7	1855	1120		
11	язык заливной	128	7	896	50	350	7	896	350	4	5
12	жульен из дичи	86	14	1204	100	1400	0	0	0	14	1
13	жульен грибной	105			100		15	1575	1500		
14	греческий	93	4	372	150	600	5	465	750	4	3

Рис. 5.35. **Импорт электронной таблицы. Определение ключа таблицы**

После этого задается имя таблицы (рис. 5.36).

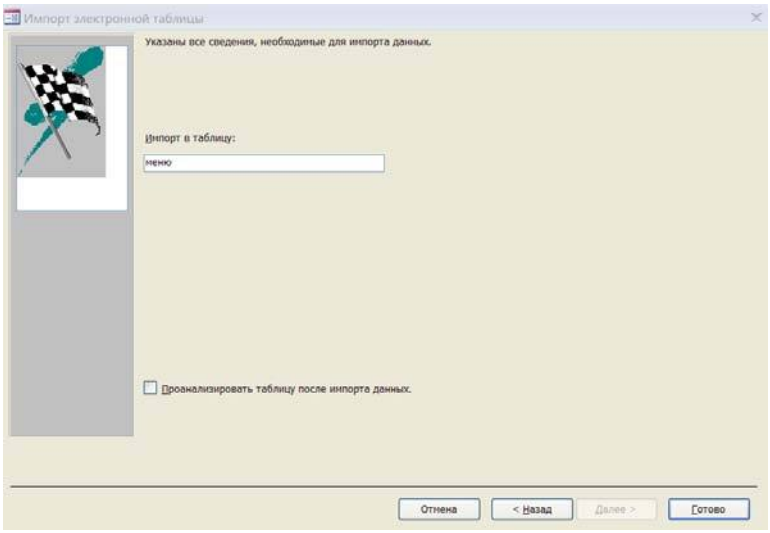


Рис. 5.36. **Импорт электронной таблицы. Определение имени таблицы**

Данные в базу данных можно импортировать и из других источников (рис. 5.37).

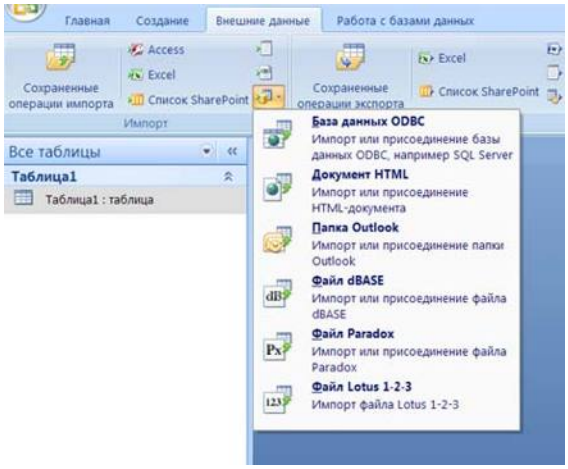


Рис. 5.37. **Импорт данных. Дополнительные источники**

Кроме того в виде таблицы можно сохранить результат запросов.

5.3. Связывание таблиц

База данных обычно включает несколько взаимосвязанных таблиц. После того, как таблицы созданы, следует задать их связанность. Для этого надо выбрать кнопку «Схема данных» на вкладке «Работа с базами данных» (рис. 5.38).

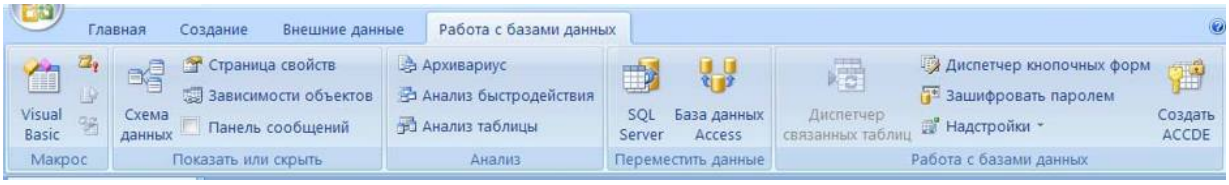


Рис. 5.38. **Вид ленты вкладки Работа с базами данных**

Далее, в открывшейся вкладке **«Схема данных»** следует добавить те таблицы, между которыми будет определяться связь. Таблицы, между которыми определяется связь, чаще всего, связаны отношением 1:М. Для установления связи надо позиционироваться на поле связи (обычно это первичный ключ) в основной таблице (та, которая стоит на стороне «1») и, не отпуская клавишу мыши, перетащить появившийся значок на соответствующее поле в «зависимом» файле и отпустить клавишу мыши. После этого на экране появится окно «**Изменение связи**» (рис. 5.39). Далее следует определить, надо 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. При связывании таблиц СОТРУДНИК и ЗНАНИЕ_ИН_ЯЗ выбрано «Обеспечение целостности данных» и «каскадное удаление связанных данных». Позиция «каскадное обновление связанных записей» не отмечена потому, что каскадное обновление означает, что при изменении первичного ключа в основной таблице соответствующие поля в связанной таблице автоматически изменяются, но в поле КОД СОТРУДНИКА в таблице СОТРУДНИК имеет тип «счетчик», а значение поля типа «счетчик» не может быть изменено.

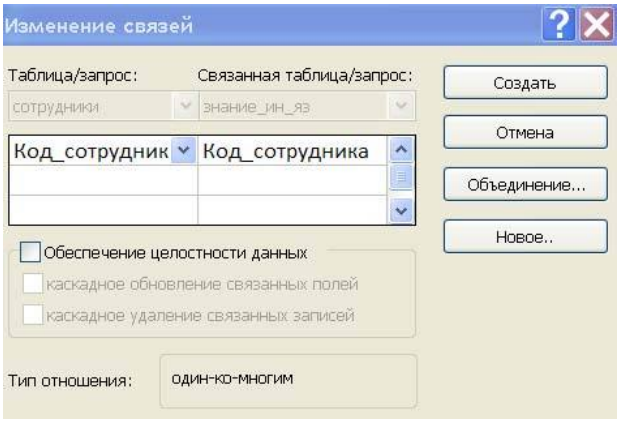


Рис. 5.39. **Задание связи и ограничений целостности по связи**

При связывании таблиц КАФЕДРА и СОТРУДНИК ограничение целостности также следует задать, чтобы в таблице СОТРУДНИК не появлялись коды кафедр, которые отсутствуют в справочнике КАФЕДРА. Задавать же «**каскадное обновление связанных полей**» в данном случае, так же как и предыдущем примере, не имеет смысла, так как в таблице КАФЕДРА поле КОД_КАФЕДРЫ имеет тип «Счетчик». Задавать каскадное удаление в данном случае опасно, так как в случае ликвидации какой-либо записи в таблице КАФЕДРА окажутся удаленными все записи сотрудников, работавших на этой кафедре.

Для связи же таблиц СОТРУДНИК и ДЕТИ каскадное удаление вполне уместно.

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.

Существуют понятия *внутреннего, левого* и *правого* соединения. В окне «Изменение **связи»**, появляющемся при установлении связи между двумя таблицами, есть кнопка «**Объединение»**, нажав на которую пользователь попадает в окно «**Параметры объединения»**, в котором он может выбрать одну из трех альтернатив:

1. объединение только тех записей, 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 совпадают;
3. объединение всех записей второй таблицы и только тех записей из первой таблицы, в которых значения связанных полей обеих таблиц совпадают.

Первая из перечисленных альтернатив обозначает *внутреннее*, вторая – *левое*, третья – *правое соединение*.

Кроме явного задания связей между таблицами можно задать возможность автообъединения (рис. 5.40). При этом связываться будут таблицы, которые имеют поля с одинаковыми именами, типами данных и длинами.

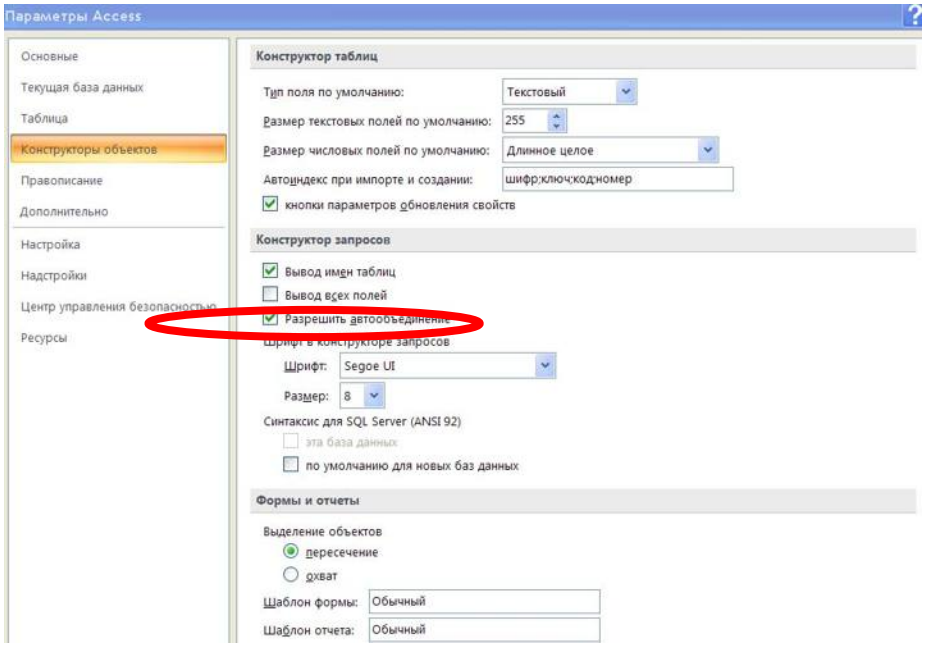


Рис. 5.40. **Настройка параметров Access**

5.4. Задание ограничений целостности

Обеспечение целостности БД является одной из важнейших задач при создании БнД, так как 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 ограничения целостности могут задаваться при создании таблицы.

Многие из свойств полей, позволяют обеспечивать контроль целостности:

- размер поля
- тип поля
- формат поля
- маска ввода
- значение по умолчанию
- условия на значения
- сообщение об ошибке
- обязательное поле
- пустые строки
- индексированное поле,

что в той или иной степени связано с ограничениями целостности.

Тип данных

Тип данных определяет допустимые символы, которые могут быть использованы при его заполнении, например не допускается ввод текста в числовые поля.

Для некоторых типов полей, например поля типа «дата», осуществляется и более сложная проверка. Если ошибочно введены недопустимые для этого типа данных симво-

лы или несуществующая дата, то пользователь должен обязательно исправить ошибку, так как иначе СУБД не даст возможность продолжить работу.

Размер поля

В поле нельзя ввести больше символов, чем это зафиксировано в свойстве «***размер поля***» или предопределено типом поля.

Условия на значения

Одной из самых гибких возможностей определения ограничений целостности является задание «**Условия на значения».** 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 года. (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 года 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 «**Условие на значение**» зафиксировано – «**нет**»).

Условия на значения могут задаваться для полей или записей. Выражения, определяющие условия на значения, не должны содержать функции, 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**» в контекстном в режиме конструктора 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. 5.41), то в сообщении об ошибке выводится текст, указанный в качестве значения этого свойства.

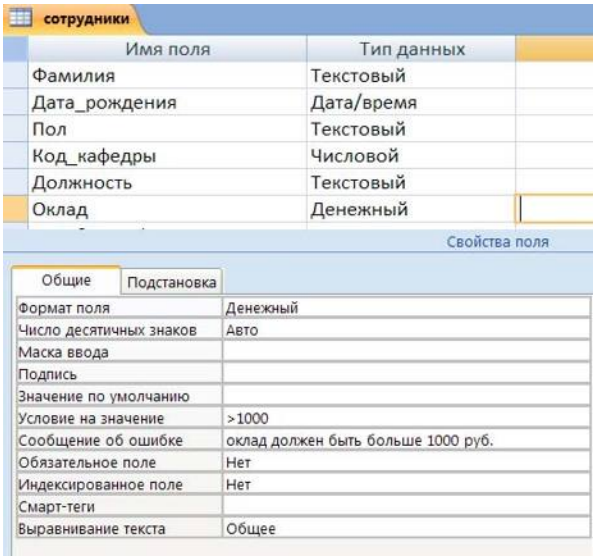


Рис. 5.41. **Задание ограничений целостности при описании полей таблицы**

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 ***Условия на значения.*** Например, для поля «Должность» в таблице БД преподавателей вузов можно задать условие «ассистент» Or «старший преподаватель» Or «доцент» Or «профессор» Or «заведующий кафедрой».

Маска ввода

Предположим, вы вводите 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 допускающую 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, которые могут быть использованы в масках, и их назначение можно посмотреть в «Справочной системе» Access.

Использование подобных масок ввода не только обеспечивает контроль использования допустимых символов, но и облегчает процесс ввода данных.

Индексированное поле

Индексированное поле можно использовать для контроля на уникальность. В Access, как и во многих других системах, при определении для индексированного поля значения свойства «*уникальный индекс*» в это поле не допускается ввод повторяющихся значений.

В тех СУБД, которые поддерживают концепцию ключа (в том числе и в Microsoft Access), после того как в таблице определяется ключ, по этому полю производится индексирование и запрещается ввод повторяющихся или пустых значений ключа.

Ограничения целостности 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:

- запретить удалять основную запись, если имеются подчиненные;
- удалить вместе с основной записью и все подчиненные (каскадное удаление).

В Access, например, при поддержании целостности связи автоматически принимается первая стратегия. Чтобы отменить ее для данной связи, надо установить флажок «***Каскадное удаление связанных записей***».

Для задания сложных условий можно использовать макросы или модули.

5.5. Ввод данных в базу данных

После того как завершено проектирование структуры базы данных, БД описана, можно приступать к вводу данных. Это можно сделать как сразу по окончании описания структуры таблицы, так и потом.

Ввод и корректировка данных в режиме «Таблица»

Как отмечалось выше, чтобы сразу после описания структуры таблицы в режиме Конструктора вводить данные в эту таблицу, надо перейти в режим таблицы. После сохранения описания таблицы, она высвечивается на экране в табличном виде (первая строка этой таблицы содержит имена полей таблицы, вторая – пустая, в которую и вводятся данные).

Для того чтобы попасть в режим «**Таблица**» для ввода данных в уже существующую таблицу, надо в области переходов позиционироваться на нужной **таблице** и **открыть** ее. Каждая таблица содержит пустую запись, которая следует за последней существующей записью и предназначена для ввода новых данных (эта запись отмечена слева символом «звездочка»(*)).

В Access для рационализации процесса ввода данных в БД можно использовать свойство поля «**Значение по умолчанию**». Свойство «Значение по умолчанию» позволяет указать значение, которое будет автоматически вводиться в поле при создании новой записи. В качестве значения по умолчанию чаще всего выбирается то значение, которое чаще всего встречается в записях БД. Например, для значения поля «Должность» в таблице, содержащей сведения о сотрудниках вуза, это будет «доцент».

Обычно в качестве значения по умолчанию указывается постоянное значение, однако можно использовать и выражение. Например: для ввода текущей даты можно ввести выражение

=Date(),

использующее функцию «Date()», выводящую текущую дату.

Если функция используется в выражении по умолчанию, то значение соответствующего поля может быть впоследствии изменено вручную.

Выражения, которые используются в качестве значений по умолчанию, не должны содержать ссылки на элементы управления и другие поля, а также функции, определенные пользователем.

Выражения могут записываться непосредственно или строиться с помощью «*Построителя выражений*».

Надо с осторожностью относиться к использованию значений по умолчанию.

Использование масок для ввода данных

Об использовании масок ввода уже немного говорилось в разделе «Создание таблиц» и «Задание ограничений целостности». Рассмотрим некоторые другие примеры. Можно использовать маски для ввода конфиденциальной информации (если использовать маску типа «пароль», то вместо символов, введенных в поле, на экране будут изображаться звездочки (*)).

Если, например, в институте принято обозначение студенческих групп, включающее две заглавные буквы, дефис и три цифры, то для этого поля можно использовать следующую маску ввода:

>LL\-000

При этом не надо будет переключаться при вводе в верхний регистр, в качестве двух первых символов можно будет ввести только буквы, а последних трех – только цифры. Знак «-» вводится и хранится в записях БД не будет, он присутствует только в маске при вводе и выводе данных.

Для ускорения ввода данных в текущее поле таблицы могут быть использованы определенные комбинации клавиш:

Таблица 5.4

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

Запись автоматически сохраняется при переходе к другой записи.

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. Что такое «ограничения целостности»?
20. Какие виды ограничений целостности вы знаете?
21. В чем важность задания ограничений целостности?
22. Что такое «ограничение целостности связи» и как они могут задаваться в Access?
23. Какие способы задания ограничений целостности в Access вы знаете?

Курсовой проект

Методические указания по написанию

Целью курсового проектирования является закрепление теоретических знаний, а также навыков проектирования БД, полученных при изучении курса «Базы данных».

Допускается написание курсовых проектов по разным категориям тем:

1. Проектирование баз данных для конкретных предметных областей.
2. Сравнительный анализ возможностей СУБД.
3. Сравнительный анализ средств автоматизации проектирования БД.
4. Создание и использование хранилищ данных.
5. Научно-исследовательские темы по любому из направлений по тематике «Базы данных».

Курсовые проекты по любой из категории тем обязательно должны включать проектную часть, выполненную на компьютере с использованием той или иной СУБД. Для выполнения курсового проекта может быть выбрана любая СУБД.

Основной группой курсовых проектов является **«Проектирование баз данных для конкретных предметных областей»**. Курсовые проекты этой группы должны содержать следующие разделы:

1. Описание предметной области. Постановка задачи.
2. Выбор средств/методологии проектирования. Выбор СУБД.
3. Построение инфологической (концептуальной) модели предметной области.
4. Проектирование логической структуры базы данных.
5. Выявление полного перечня ограничений целостности, присущего данной предметной области. Определение перечня ограничений целостности, которые будут контролироваться в данном курсовом проекте. Выбор способа реализации контроля целостности для каждого из ограничений.
6. Проектирование физической структуры базы данных.
7. Организация ввода данных в БД.
8. Организация корректировки БД.
9. Описание информационных потребностей пользователей и выбор способов их реализации.
10. Разработка интерфейса.
11. Реализация проекта в среде конкретной СУБД.

Раздел *«Описание предметной области. Постановка задачи»* должен содержать всю необходимую и достаточную информацию для проектирования Базы данных. Прежде всего, должен быть определен круг лиц, который будет иметь доступ к базе данных, их права и обязанности. Описаны бизнес-процессы, происходящие в предметной области. Приведены формы всех входных и выходных документов, описаны регламентированные запросы. Определена периодичность решения всех задач. Описаны алгоритмы получения промежуточных и результатных показателей, изображен граф взаимосвязей показателей.

Предметная область должна быть описана с такой степенью подробности, чтобы можно было определить характер связи между объектами. Так, например, если предметной областью является вуз, необходимо рассмотреть следующие ситуации. Если, например, для студента разрешен экстернат, возможно ли, что он не приписан ни к одной студенческой группе? Кроме того, некоторые вузы бывают полностью заочными. Также надо определить, может ли один и тот же студент одновременно получать несколько специ-

альностей одновременно в одном и том же вузе. В вузе могут быть дистанционные формы обучения. Необходимо уточнить, могут ли студенты, обучающиеся полностью дистанционно, обучаться на бюджетной основе или это не предусмотрено современным законодательством. И таких нюансов много, и все они должны быть выявлены и описаны.

Вуз может быть чисто коммерческий, где все студенты обучаются на платной основе, а может финансироваться на смешанной основе. В последнем случае надо знать, как формируются группы.

Описывая любую организацию, надо оговорить, допустимо или нет внутреннее совместительство.

Описывая предметную область, надо знать действующее законодательство. Эта информация позволит определить, какие свойства необходимо определить для объекта, а также выявить ограничения целостности. Так, например, на дневное отделение вузов можно принимать лиц в возрасте до 35 лет. Стипендию можно платить, если студент получает первое образование. Есть ограничения на минимально допустимую зарплату, на минимальный возраст, с которого можно принимать на работу.

В разделе «*Выбор средств/методологии проектирования. Выбор СУБД*» должна быть выбрана методика проектирования базы данных. СУБД также может быть выбрана любая. Степень подробности описания тех средств, которые рассматриваются при выборе, зависит от того, насколько распространенные средства используются при анализе. Если известные и описанные в используемой в учебном процессе литературе, то подробно описывать их не надо, надо просто обосновать свой выбор. В противном случае описание должно быть более подробным.

В разделе «*Построение инфологической (концептуальной) модели предметной области*» надо построить ER-модель в выбранной в разделе 2 нотации. Рекомендуется сначала построить базовую ER-модель, а затем ER-модель в среде любой CASE-системы. ER-модель должна полностью соответствовать описанию предметной области, приведенному в разделе 1.

Содержание раздела «*Проектирование логической структуры базы данных*» будет зависеть от выбранной СУБД, методики проектирования и, если используются средства автоматизации проектирования, особенностей выбранного средства проектирования. Если алгоритм проектирования логической структуры базы данных – многовариантный, то необходимо мотивировать выбор варианта. Кроме того, на этапе логического проектирования необходимо определить типы и длины полей. В данном разделе должна быть распечатана созданная схема базы данных.

Выявление полного перечня ограничений целостности, присущего данной предметной области, осуществляется, во-первых, на основе информации из раздела «Описание предметной области». Далее должны быть выявлены ограничения целостности, вызванные особенностями используемой СУБД. Не все выявленные ограничения должны контролироваться в БД. Необходимо определить перечень ограничений целостности, которые будут контролироваться в данном КП. Далее необходимо выбрать способ реализации контроля целостности для каждого из ограничений. Необходимо не просто описать выбранный способ, но и привести соответствующие распечатки, отражающие их реализацию в конкретной системе.

Проектирование физической структуры базы данных существенно зависит от выбранной СУБД.

В разделе «*Организация ввода данных в БД*» должны быть разработаны экранные формы ввода данных. Организация корректировки БД может потребовать разработку специальных форм для выполнения тех или иных видов корректировки.

В разделе «*Реализация запросов, получение отчетов*» необходимо реализовать запросы и получить отчеты, описанные в разделе «Описание предметной области».

Результатом выполнения раздела «*Разработка интерфейса*» должна быть разработанная и реализованная система меню, отражающая весь функционал системы. Система меню должна отражать те функции, которые выполняют пользователи базы данных.

Работа спроектированной системы должна быть опробована на контрольном примере.

Курсовые проекты по проектированию баз данных могут быть выполнены для любой предметной области.

Помимо предлагаемых нами тем студенты могут выбрать свою предметную область.

Сравнительный анализ возможностей СУБД. Курсовые проекты данной группы могут касаться любых функциональных возможностей СУБД. Так как СУБД являются сложными многофункциональными системами, то рекомендуется выбирать тему по какой-либо одной из функций:

- генераторы запросов
- генераторы отчетов
- генераторы экранных форм
- генераторы приложений
- обеспечение безопасности
- обеспечение целостности
- работа в многопользовательском режиме
- реплицирование

и т.п.

По выбранной для сравнения функции необходимо дать ее описание, произвести классификацию предоставляемых возможностей, определить критерии для сравнения.

Для сравнения должны быть выбраны две или более СУБД из числа доступных студенту систем. Курсовой проект должен включать реализацию на конкретных примерах возможностей каждой из выбранных СУБД.

Сравнительный анализ средств автоматизации проектирования БД. Курсовые проекты данной группы должны включать описание сущности автоматизированного проектирования БД, существующих методологий, критериев сравнения CASE-средств.

Для сравнения должны быть выбраны две или более CASE-системы из числа доступных студенту систем. Курсовой проект должен включать описание предметной области в среде выбранных CASE-систем и получение сгенерированных проектных решений, реализацию на конкретных примерах возможностей каждой из выбранных CASE-систем.

Создание и использование хранилищ данных. Курсовые проекты данной группы должны включать теоретическую часть, отражающую проблематику хранилищ данных, и проектную часть, выполненную в конкретной программно-технологической среде.

Курсовые проекты данной группы могут включать как вопросы создания и использования хранилищ данных, так и только использование ранее созданных хранилищ, если студент имеет доступ к таким хранилищам.

Научно-исследовательские темы по любому из направлений по тематике «Базы данных». Содержание курсовых проектов данной группы будет зависеть от выбранной темы исследования. Курсовой проект должен содержать постановку проблемы, описание существующего состояния, пути решения проблемы. Желательно, чтобы работа включала собственный вариант решения проблемы и, по возможности, его реализацию на ЭВМ.

Примерные темы курсовых

1. Проектирование БД для работника склада (варианты: склад торговой организации, занимающейся продажей как продукции собственного производства, так и продукции внешних поставщиков; склад оптовой торговой организации; склад готовой продукции; склад сырья и материалов и др.).
2. Проектирование БД для контроля выполнения нагрузки преподавателей вуза.
3. Проектирование БД для контроля сессионной успеваемости студентов вуза.
4. Проектирование БД для учета контингента студентов вуза.
5. Проектирование БД для организации дипломного проектирования в вузе.
6. Проектирование БД для организации курсового проектирования.
7. Проектирование БД для профкома вуза.
8. Проектирование БД для начисления стипендии в вузе.
9. Проектирование БД для библиотеки вуза.
10. Проектирование БД для управления работой компьютерных аудиторий учебного заведения.
11. Проектирование БД для управления работой класса свободного доступа.
12. Проектирование БД для начисления заработной платы преподавателей.
13. Проектирование базы данных Ученого совета по защите диссертаций.
14. Проектирование базы данных Отдела аспирантуры.
15. Проектирование БД для контроля успеваемости школьников.
16. Проектирование БД детского сада.
17. Проектирование БД спортивной школы.
18. Проектирование БД центра детского творчества.
19. Проектирование БД партнеров софтверной фирмы.
20. Проектирование БД коммерческого учебного центра.
21. Проектирование БД для расчета заработной платы (варианты: преподавателей вуза, всех сотрудников вуза, предприятий/организаций с разными системами оплаты труда).
22. Проектирование БД для учета домашних финансов.
23. Проектирование БД для домашней библиотеки.
24. Проектирование БД для районной библиотеки.
25. Проектирование БД для домашней видеотеки.
26. Проектирование БД для пункта проката видеофильмов.
27. Проектирование БД кинотеатра.
28. Проектирование БД драматического театра.
29. Проектирование БД для домашней аудиотеки.
30. Проектирование БД тренера спортивной команды.
31. Проектирование БД агентства по аренде квартир.
32. Проектирование БД риэлтерского агентства.
33. Проектирование БД для учета услуг, оказываемых юридической консультационной фирмой.

34. Проектирование БД для автосервисной фирмы.
35. Проектирование БД для автозаправочной станции.
36. Проектирование БД центра по продаже автомобилей.
37. Проектирование БД таксомоторного парка.
38. Проектирование БД по подсистеме «Кадры» (варианты: для вуза, школы, промышленного предприятия, торговой фирмы, софтверной фирмы и т.п.).
39. Проектирование БД службы знакомств.
40. Проектирование базы данных туристического агентства.
41. Проектирование базы данных туристического оператора.
42. Проектирование базы данных туристического клуба.
43. Проектирование БД районной поликлиники. Подсистема «Работа с пациентами».
44. Проектирование БД районной поликлиники. Подсистема «Учет льготных лекарств».
45. Проектирование БД районной поликлиники. Подсистема «Планирование и учет работы медицинского персонала».
46. Проектирование БД районной поликлиники. Подсистема «Учет пациентов».
47. Проектирование базы данных родильного дома.
48. Проектирование базы данных больницы. Подсистема «Работа с пациентами».
49. Проектирование базы данных больницы. Подсистема «Лекарственное обеспечение».
50. Проектирование базы данных аптеки.
51. Проектирование базы данных гостиницы. Подсистема «Работа с клиентами».
52. Проектирование базы данных дачного кооператива.
53. Проектирование базы данных издательства. Подсистема «Работа с авторами».
54. Проектирование базы данных издательства. Подсистема «Служба маркетинга».
55. Проектирование базы данных Учета расчетов с клиентами в банке.
56. Проектирование базы данных строительной фирмы.
57. Проектирование базы данных городской телефонной сети. Подсистема «Учет расчетов с клиентами».
58. Проектирование базы данных торговой организации.
59. Проектирование базы данных аэропорта.
60. Проектирование базы данных ГИБДД.
61. Проектирование базы данных фотоцентра.
62. Проектирование базы данных горнолыжной базы.
63. Проектирование базы данных ателье верхней одежды.
64. Проектирование базы данных телеателье.
65. Проектирование базы данных пункта по ремонту электроаппаратуры.
66. Проектирование БД для пункта проката автомобилей.

Список рекомендуемой литературы

1. Martin James. Fourth-generation languages. – Vol. 1. – New Jersy: Prentice-Hall, Inc., 1989.
2. Hansen Gary W., Hansen James V. Database Management and Design. – New Jersy: Prentice-Hall, Inc., 1992.
3. Вендров А. М. Case-технологии. Современные методы и средства проектирования информационных систем. – М.: Финансы и статистика, 1988.
4. Дейт К.Дж. Введение в системы баз данных / Пер. с англ. – 6-е изд. – СПб.: Издательский дом «Вильямс», 2000.
5. Джексон Г. Проектирование реляционных баз данных для использования с микро-ЭВМ / Пер. с англ. – М.: Мир, 1991.
6. Диго С.М. Базы данных: проектирование и использование: Учебник. – М.: Финансы и статистика, 2005.
7. Диго С.М. Проектирование баз данных: Учебник. – М.: Финансы и статистика, 1988.
8. Калянов Г.Н. CASE – структурный системный анализ. – М.: ЛОРИ, 1996.
9. Когаловский М.Р. Энциклопедия технологий баз данных. – М.: Финансы и статистика, 2002.
10. Маклаков С.В. Создание информационных систем с ALLFusion Modeling Suite. – М.: ДИАЛОГ-МИФИ, 2005 – 432 с.
11. Мишенин А.И. Теория экономических информационных систем. – М.: Финансы и статистика, 2003.
12. О правовой охране программ для электронных вычислительных машин и баз данных. Закон №3523-1, 23.09.92.
13. Общеотраслевые руководящие материалы по созданию банков данных. – М.: ГКНТ, 1982.
14. Саймон А.Р. Стратегические технологии баз данных: менеджмент на 2000 год. – М.: Финансы и статистика, 1998.
15. Фаулер М., Скотт К. UML в кратком изложении. Применение стандартного языка объектного моделирования / Пер. с англ. – М.: Мир, 1999.
16. Фаулер М. UML. Основы / Пер. с англ. – 3-е изд. – СПб: Символ-Плюс, 2004. – 192 с., ил.
17. Хансен Г., Хансен Дж. Базы данных. Разработка и управление. – М.: Бином, 1999.
18. Четвериков В.Н., Ревунков Г.И., Самохвалов Э.Н. Базы и банки данных. – М.: Высшая школа, 1987.