Содержание

Введение в базы данных

Современные технологии баз данных являются одним из существенных факторов успеха в любом виде человеческой деятельности, обеспечивая хранение корпоративной информации, представление данных для пользователей и клиентов в среде World Wide Web и поддержку многих других процессов. Помимо того, базы данных составляют основу разнообразных научных проектов. Они позволяют накапливать информацию, собранную инженерами, исследователями генотипа человека, биохимиками и специалистами многих других отраслей знания.

Мощь баз данных основывается на результатах исследований и технологических разработок, полученных на протяжении нескольких последних десятилетий, и заключена в специализированных программных продуктах, которые принято называть ***системами управления базами данных*** (СУБД) (*Database Management Systems — DBMS*). СУБД — это эффективный инструмент сбора больших порций информации и действенного управления ими, позволяющий сохранять данные в целости и безопасности на протяжении длительного времени. СУБД относятся к категории наиболее сложных программных продуктов, имеющихся на рынке в настоящее время. СУБД предлагают пользователям следующие функциональные возможности.

1. *Средства постоянного хранения данных.* СУБД, подобно файловым системам, поддерживают возможности хранения чрезвычайно больших фрагментов данных, которые существуют независимо от каких бы то ни было процессов их использования. СУБД, однако, значительно превосходят файловые системы в отношении гибкости представления информации, предлагая структуры, обеспечивающие эффективный доступ к большим порциям данных.

2. *Интерфейс программирования.* СУБД позволяет пользователю или прикладной программе обращаться к данным и изменять их посредством команд развитого языка запросов. Преимущества СУБД по сравнению с файловыми системами проявляются и в том, что первые дают возможность манипулировать данными самыми разнообразными способами, гораздо более гибкими, нежели обычные операции чтения и записи файлов.

3. *Управление транзакциями.* СУБД поддерживают параллельный доступ к данным, т.е. возможность единовременного обращения к одной и той же порции данных со стороны нескольких различных процессов, называемых ***транзакциями***(transactions). Чтобы избежать некоторых нежелательных последствий подобного обращения, СУБД реализуют механизмы обеспечения ***изолированности***(*isolation*) транзакций (транзакции выполняются независимо одна от другой, как если бы они активизировались строго последовательно), их ***атомарности*** (*atomicity*) (каждая транзакция либо выполняется целиком, либо не выполняется вовсе) и ***устойчивости*** (*durability*) (системы содержат средства надежного сохранения результатов выполнения транзакций и самовосстановления после различного рода ошибок и сбоев).

База данных по существу не что иное, как набор порций информации, существующий в течение длительного периода времени (возможно, исчисляемого годами или даже десятилетиями). Термином ***база данных*** (*database*) в соответствии с принятой традицией обозначают набор данных, находящийся под контролем СУБД. Добротная СУБД обязана обеспечить реализацию следующих требований.

1. Позволять пользователям создавать новые базы данных и определять их *схемы* (логические структуры данных) с помощью некоторого специализированного языка, называемого ***языком определения данных*** (*Data Definition Language — DDL*).

2. Предлагать пользователям возможности задания ***запросов*** (*queries*) и модификации данных средствами соответствующего ***языка запросов*** (*query language*), или ***языка управления данными*** (*Data Manipulation Language — DML*).

3. Поддерживать способность сохранения больших объемов информации — до многих гигабайтов и более — на протяжении длительных периодов времени, предотвращая опасность несанкционированного доступа к данным и гарантируя эффективность операций их просмотра и изменения.

4. Управлять единовременным доступом к данным со стороны многих пользователей, исключая возможность влияния действий одного пользователя на результаты, получаемые другим, и запрещая совместное обращение к данным, чреватое их порчей.

Появление первых коммерческих систем управления базами данных относится к концу 1960-х годов. Непосредственными предшественниками таких систем были файловые системы, удовлетворявшие только некоторым из требований, перечисленных выше (см. п. 3). Файловые системы действительно пригодны для хранения обширных фрагментов данных в течение длительного времени, но они не способны гарантировать, что данные, не подвергшиеся резервному копированию, не будут испорчены или утеряны, и не поддерживают эффективные инструменты доступа к элементам данных, положение которых в определенном файле заранее не известно.

Файловые системы не отвечают и условиям п. 2, т.е. не предлагают языка запросов, подходящего для обращения к данным внутри файла. Обеспечиваемая ими реализация требований п. 1, предполагающих наличие возможности создания схем данных, ограничена способностью определения структур каталогов. Наконец, файловые системы совершенно не удовлетворяют условиям п. 4. Даже если система и не препятствует параллельному доступу к файлу со стороны нескольких пользователей или процессов, она не в состоянии предотвратить возникновение ситуаций, когда, например, два пользователя в один и тот же момент времени пытаются внести изменения в один файл: результаты действий одного из них наверняка будут утрачены.

К числу первых серьезных программных приложений СУБД относились те, в которых предполагалось, что данные состоят из большого числа элементов малого объема; для обработки таких элементов требовалось выполнять

множество элементарных запросов и операций модификации. Кратко рассмотрим одну из таких приложений баз данных.

Система бронирования авиабилетов

Система подобного типа имеет дело со следующими элементами данных:

1) сведениями о резервировании конкретным пассажиром места на определенный авиарейс, вылетающий в определенную дату, включая информацию о номере места и обеденном меню;

2) информацией о полетах — аэропортах отправления и назначения для каждого рейса, сроках отправки и прибытия, принадлежности воздушных судов тем или иным компаниям, типам самолетов, количеству мест и т.п.;

3) данными о ценах на авиабилеты, поступивших заявках и наличии свободных мест.

В типичных запросах требуется выяснить, какие рейсы из одного заданного аэропорта в другой близки по времени отправления к указанному календарному периоду, имеются ли свободные места и какова стоимость билетов. К числу характерных операций по изменению данных относятся бронирование места на рейс, определение номера места и выбор обеденного меню. В любой момент к одним и тем же элементам данных могут обращаться несколько операторов-кассиров. СУБД обязана обеспечить подобную возможность, но исключить любые потенциальные проблемы, связанные, например, с продажей нескольких билетов на одно место, а также предотвратить опасность потери записей данных, если система внезапно выйдет из строя.

Самые первые СУБД, во многом унаследовавшие свойства файловых систем, оказывались способными представлять результаты запросов практически только в том виде, который соответствовал структуре хранения данных. В подобных системах баз данных находили применение несколько различных моделей описания структуры информации — в основном, "иерархическая" древовидная и "сетевая" графовая. В конце 1960-х годов последняя получила признание и нашла отражение в стандарте CODASYL (Committee on Data Systems and Languages).

Одной из основных проблем, препятствовавших распространению и использованию таких моделей и систем, было отсутствие поддержки ими высокоуровневых языков запросов. Например, язык запросов CODASYL для перехода от одного элемента данных к другому предусматривал необходимость просмотра вершин графа, представляющего элементы и связи между ними. Совершенно очевидно, что в таких условиях для создания даже самых простых запросов требовались весьма серьезные усилия.

Системы реляционных баз данных

После опубликования в 1970 году статьи Э.Ф. Кодда (E.F. Codd) системы баз данных претерпели существенные изменения. Д-р Кодд предложил схему представления данных в виде таблиц, называемых ***отношениями*** (*relations*). Структуры таблиц могут быть весьма сложными, что не снижает скорости обработки самых различных запросов. В отличие от ранних систем баз данных, рассмотренных выше, пользователю реляционной базы данных вовсе не требуется

знать об особенностях организации хранения информации на носителе. Запросы к такой базе данных выражаются средствами высокоуровневого языка, позволяющего значительно повысить эффективность работы программиста.

Запросы к базе данных выполняются на языке **SQL** (***Structured Query Language — язык структурированных запросов***) — наиболее важном и мощном представителе семейства языков запросов.

Размеры корпоративных баз данных зачастую исчисляются сотнями гигабайтов. Помимо того, с удешевлением носителей информации человек получает в свое распоряжение новые возможности хранения все возрастающих массивов данных.

Помимо того, современные базы данных способны хранить не только простые элементы данных, такие как целые числа и короткие строки символов. Они позволяют накапливать графические изображения, аудио- и видеозаписи и чрезвычайно крупные фрагменты данных других типов. Например, для хранения видеоматериалов часовой продолжительности воспроизведения требуется участок носителя емкостью в один гигабайт.

Способность системы сохранять громадные объемы данных, разумеется, важна, но она вряд ли будет востребована, если производительность такой системы недостаточно высока. Поэтому СУБД, обрабатывающие чрезмерно большие фрагменты информации, нуждаются в реализации решений, позволяющих повысить скорость вычислений. Один из важнейших подходов к проблеме связан с использованием структур ***индексов*** (*indexes*) —Другой способ достижения поставленной цели — обработать больше данных в течение промежутка времени заданной продолжительности — предполагает обращение к средствам распараллеливания вычислений. Параллелизм вычислений реализуется в нескольких направлениях.

Современное программное обеспечение поддерживает архитектуру ***клиент/сервер*** (*client/server*), в соответствии с которой запросы, сформированные одним процессом *(клиентом),* отсылаются для обработки другому процессу *(серверу)*. Системы баз данных в этом смысле — не исключение: обычной практикой является разделение функций СУБД между процессом сервера и одним или несколькими клиентскими процессами.

Простейший вариант архитектуры "клиент/сервер" предполагает, что СУБД целиком представляет собой сервер, за исключением интерфейсов запросов, которые взаимодействуют с пользователем и отсылают запросы и другие команды на сервер с целью их выполнения. Например, в реляционных системах для представления запросов клиентов к серверу используются инструкции языка SQL. Результаты выполнения запросов сервером возвращаются клиентам в форме таблиц, или отношений. Взаимосвязь клиента и сервера может быть гораздо более сложной, особенно в тех случаях, когда результаты выполнения запросов обладают сложной структурой или большим объемом.

В последнее время широкое распространение получили многоуровневые архитектуры, в которых СУБД отводится роль поставщика динамически генерируемого содержимого Web-сайтов. СУБД продолжает действовать как

сервер базы данных, но его клиентом теперь служит ***сервер приложений*** (*application server*), который управляет подключениями к базе данных, транзакциями, авторизацией и иными процессами. Клиентами серверов приложений в свою очередь могут являться Web-серверы, обслуживающие конечных пользователей, и другие программные приложения.

1. Модель данных "сущность-связь"

Процесс проектирования базы данных начинается с анализа того, какого рода документы, справки или отчеты мы должны получать с ее помощью. Второй шаг проектирования – это определение того, какая информация должна быть представлена в базе данных и каковы взаимосвязи между элементами этой информации. Структура, или ***схема***, базы данных определяется средствами различных языков или систем обозначений, пригодных для описания проектов. По завершении этапа уточнений и согласований проект преобразуется в форму, которая может быть воспринята СУБД.

Один из таких подходов построения базы данных – это так называемая ***ER-модель,*** или ***модель "сущность—связь "*** (*Entity-Relationship model*), ставшая традиционной и наиболее популярной. По своей природе модель является графической: прямоугольники отображают элементы данных, а линии (возможно, со стрелками) указывают на связи между ними.

Модель проста и является основой для реализации базы данных на большинстве современных СУБД. Процесс проектирования базы данных обычно начинается с разработки ее схемы посредством инструментов, предлагаемых ER-моделью, либо некоторой объектной моделью с последующей трансляцией схемы в реляционную модель, подлежащую физической реализации.

Рис. 1.1 иллюстрирует, каким образом ER-модели используются при проектировании баз данных. Обычно проектирование принято начинать с изучения *понятий и описаний* информации, подлежащей моделированию, а затем пытаться отобразить их в рамках ER-модели. Затем *ER-проект* преобразуется в *реляционную схему,* выраженную средствами языка определения данных для конкретной СУБД. В большинстве случаев СУБД основываются на реляционной модели. Если дело обстоит именно так, в ходе довольно прямолинейного процесса, ER-диаграмма обретает конкретную форму, называемую *реляционной схемой базы данных* (*relational database schema*).

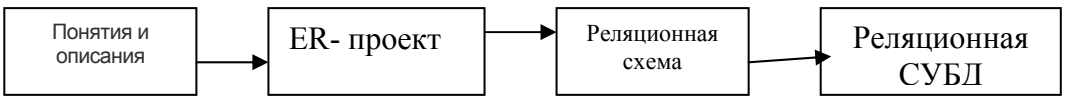


Рис. 1.1. Процесс моделирования и реализации базы данных

Важно отметить, что в то время как в СУБД подчас находят применение модели, отличные от реляционных или объектно-реляционных, систем баз данных, способных реализовать ER-модель непосредственно, попросту не

существует. Причина заключается в том, что эта модель недостаточно удовлетворительно согласовывается с эффективными структурами данных, на основе которых должна создаваться "реальная" база данных.

Элементы ER-модели

Наиболее распространенным средством абстрактного представления структур баз данных является ***ER-модель***, или ***модель "сущность—связь"*** (*entity-relationship model*). В ER-модели структура данных отображается графически, в виде *диаграммы сущностей и связей*, состоящей из элементов трех основных типов:

a) *множеств сущностей;*
b) *атрибутов (свойств сущности),*
c) *связей.*

Множества сущностей

Сущности

Сущность (*entity*) — это абстрактный объект определенного вида. Набор однородных сущностей образует ***множество сущностей*** (*entity set*). Понятие сущности обладает определенным сходством с понятием ***объекта*** (*object*) (если трактовать последнее так, как это принято делать в объектно-ориентированном проектировании). Примерно таким же образом соотносятся множество сущностей и ***класс объектов***. ER-модель, однако, отображает ***статические*** объекты — она имеет дело со *структурами* данных, но не с *операциями* над данными. Поэтому предполагать, что в ней могут содержаться описания неких "методов", соответствующих множествам сущностей и аналогичных методам класса, нет никаких оснований.

Пример 1.1. Мы будем рассматривать и развивать пример, касающийся базы данных о книгах, написавших их авторах, издательствах, их издававших, и т.п. Каждая из книг представляет собой сущность, а набор всех книг образует множество сущностей. Авторы, написавшие книги, также являются сущностями, но другого вида, и их множество — это множество сущностей. Издательство — это сущность еще одного вида, а перечень издательств формирует третье множество сущностей, которое также будет использоваться в дальнейшем.

Атрибуты

Множеству сущностей отвечает набор ***атрибутов*** (*attributes*), являющихся свойствами сущностей множества. Например, множеству сущностей ***Книги*** могут быть поставлены в соответствие такие атрибуты, как ***название*** и ***объем*** (количество страниц). В ER-модели мы предполагаем, что атрибуты представляют собой атомарные значения (например, строки, целые или вещественные числа и т.д.). Существуют и такие варианты модели, в которых понятие типа атрибута трактуется иным образом

Связи

Связи (*relationships*) — это соединения между двумя или большим числом множеств сущностей. Если, например, ***Книги*** и ***Авторы*** — это два множества сущностей, вполне закономерно наличие связи ***Пишут***, которая соединяет множества сущностей ***Книги*** и ***Авторы,*** сущность *m* множества ***Книги*** соединена с сущностью *s* множества ***Авторы*** посредством связи ***Пишут,*** если автор *s* написал книгу *m*. Мы будем рассматривать только ***бинарные связи*** (binary relationships), соединяющие *два* множества сущностей, хотя ER-модель допускает наличие связей, охватывающих произвольное количество множеств сущностей.

Диаграммы сущностей и связей

Диаграмма сущностей и связей (*entity - relationship diagram*), или ***ER-диаграмма*** (ER-diagram), — это графическое представление *множеств сущностей,* их *атрибутов* и *связей*. Элементы названных видов описываются вершинами графа, и для задания принадлежности элемента к определенному виду используется специальная геометрическая фигура:

- *прямоугольник* — для множеств сущностей;
- *овал* — для атрибутов;
- *ромб* — для связей.

Ребра графа соединяют множества сущностей с атрибутами и служат для представления связей между множествами сущностей.

Пример 1.2. На рис. 1.2 приведена **ER-диаграмма,** представляющая структуру простой базы данных, содержащей информацию о книгах. В составе диаграммы имеется три множества сущностей: *Книги*, *Авторы* и *Издательства* .

Множество сущностей *Книги* обладает четырьмя атрибутами: *Название*, *Год*, *Объем и Тип*) - "У" ("учебник"), М («мелодрама») или "Д" ("детектив"). Два других множества сущностей, *Авторы* и *Издательства,* содержат по паре однотипных атрибутов, *Фамилия* или *Название* и *Адрес*. На диаграмме представлены две связи, описанные ниже.

Рисунок 1.2. ER-модель

1. ***Пишут*** - это связь, соединяющая каждую сущность ***Книги*** с сущностями ***Авторы***, написавшими книгу. Связь ***Пишут,*** рассматриваемая в противоположном направлении, в свою очередь соединяет авторов с книгами.

2. Связь ***Публикуют*** соединяет каждую сущность ***Книги*** с сущностью ***Издательства***, выпустившей книгу. Стрелка, задающая направление от связи ***Публикуют*** к множеству сущностей ***Издательство,*** свидетельствует о том, что каждая книга издана в одном ***Издательстве***.

Экземпляры ER- диаграммы

ER-диаграммы представляют собой инструмент описания *схем*, или структур, баз данных. Базу данных, соответствующую определенной ER-диаграмме и содержащую конкретный набор данных, принято называть ***экземпляром*** базы данных (*database instance*). Каждому множеству сущностей в экземпляре базы данных отвечает некоторый частный конечный набор сущностей, а каждая из таких сущностей обладает определенными значениями каждого из атрибутов. Уместно отметить, что информация о сущностях, атрибутах и связях носит абстрактный характер: содержимое ER-модели не может быть сохранено в базе данных непосредственно. Однако представление о том, что такие данные будто бы реально существуют, помогает на начальной стадии проекта — пока мы не перейдем к отношениям и структуры данных не приобретут физическую форму.

Зачастую оказывается полезным представлять множество данных связи в виде таблицы. Столбцы этой таблицы озаглавлены наименованиями множеств сущностей, охватываемых связью, а каждому списку соединенных сущностей отводится одна строка таблицы.

Пример. 1.3. Экземпляр связи ***Пишут*** легко описать таблицей пар данных, которая может иметь следующий вид:

Книги	***Авторы***
Системы баз данных	Джефри Ульман
Системы баз данных	Дженнифер Уидом
Теория Компиляции	Джефри Ульман

Члены множества данных связи — это строки таблицы. Например, Системы баз данных Джефри Ульман представляет собой кортеж множества данных для текущего экземпляра связи ***Пишут***.

Множественность бинарных связей

Бинарная связь (*binary relationship*) в общем случае способна соединять любой член одного множества сущностей с любым членом другого множества сущностей. Однако весьма распространены ситуации, в которых свойство "множественности" связи некоторым образом ограничивается. Предположим,

что *R* - связь, соединяющая множества сущностей *E* и *F*. Тогда возможно выполнение одного из нескольких условий, перечисленных ниже.

• Если каждый член множества *E* посредством связи *R* может быть соединен не более чем с одним членом F, принято говорить, что *R* представляет связь типа ***многие к одному*** (*many-to-one relationship*), направленную от *E* к *F*. В этом случае каждая сущность множества *F* допускает соединение с многими членами *E*. Если же член *F* посредством связи *R* может быть соединен не более чем с одним членом *E,* мы говорим, что *R* — это связь "многие к одному", направленная от *F к E* (или, что то же самое, связь типа ***один ко многим*** (*one-to-many relationship*), направленная от *E* к *F).*

• Если связь *R в* обоих направлениях, от *E* к *F* и от *F* к E, относится к типу "многие к одному", говорят, что *R*— это связь типа *"один к одному"* (*one-to-one relationship*). В этом случае каждый элемент одного множества сущностей допускает соединение не более чем с одним элементом другого множества сущностей.

• Если связь *R* ни в одном из направлений — ни от *E* к *F* и ни от *F* к *E* — не относится к типу "многие к одному", имеет место связь типа ***"многие ко многим"***(*many-to-many relationship*).

Как мы уже отмечали в примере 2.2, стрелки в ER-диаграмме используются для отображения факта множественности связей. Если связь относится к типу "многие к одному" и соединяет множество сущностей *E* с множеством сущностей F, она отображается в виде стрелки, направленной к *F*. Стрелка указывает, что каждая из сущностей множества E связана не более чем с одной сущностью множества F. Если при этом линия не снабжена противоположной стрелкой, обращенной к E, сущность множества F допускает связь со многими сущностями множества E.

Пример 1.4. Если следовать рассмотренной логике, связь типа "один к одному" между множествами сущностей E и F должна представляться на диаграмме двунаправленной стрелкой, один конец которой обращен в сторону множества E, а другой — в сторону F. На рис. 1.3 показаны два множества сущностей, *Университет* и *Ректор*, соединенные связью *Возглавляет* (атрибуты сущностей для краткости опущены). Уместно предположить, что каждый ректор вправе руководить только одним университетом, а каждый университет может возглавляться только одним ректором. Поэтому связь *Управляет* следует отнести к типу "один к одному" и соединить на диаграмме с множествами сущностей *Университет* и *Ректор* посредством двух стрелок, по одной на каждое множество (так, как показано на рис.1.3)

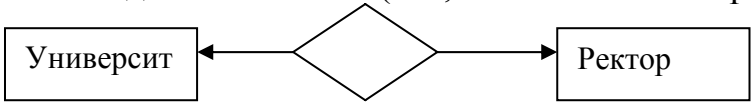


Рис. 1.3. Связь типа "один к одному"

Следует помнить: стрелка означает, что в связи участвует "не более чем один" элемент множества сущностей, на которое она указывает. При этом

обязательное наличие такого элемента в составе множества не гарантируется. Рассматривая рис. 4, мы вправе полагать, что некий "ректор" обязательно связан с определенным университетом — иначе на каком основании он мог бы величать себя ректором? Однако университет в какой-то период времени может обходиться без руководителя, так что стрелка, направленная от *Управляет* к *Ректор*, на самом деле означает именно "не более чем один", но не "в точности один.

2. Принципы проектирования

Целесообразно обсудить ряд принципиальных вопросов модели "сущность-связь", касающихся того, *что* представляет собой хороший проект базы данных и *чего* в процессе дизайна желательно избегать. В этом разделе мы предложим вашему вниманию сведения об основных принципах проектирования.

Достоверность

Прежде всего, проект обязан достоверно представлять спецификацию базы данных. Другими словами, множества сущностей и их атрибуты должны соответствовать реальным требованиям. Вы не вправе, например, ассоциировать с множеством сущностей *Авторы* атрибут *количество цилиндров*, хотя последний вполне применим по отношению к множеству *Автомобили*. Прежде чем вводить в модель новое множество сущностей, атрибут или связь, надлежит задаться вопросом, а все ли известно о той части реального мира, которая должна быть смоделирована.

Пример 2.1. Если вновь присмотреться к связи *Пишут,* соединяющей множества сущностей *Авторы* и *Книги*, станет очевидным, что она должна относиться к типу "многие ко многим". Причины бесспорны: здравый смысл, основанный на элементарном понимании моделируемых сущностей, подсказывает, что авторы вправе писать несколько книг, а в каждую книгу могут писать несколько авторов. Совершенно неправомерно считать, что связь *Пишут* в любом направлении может представлять отношение "многие к одному" или, более того, "один к одному".

Пример 2.2. Подчас, однако, далеко не очевидно, как именно те или иные объекты реального мира должны соотноситься в рамках ER-модели. Рассмотрим для примера множества сущностей *Учебные курсы* и *Преподаватели*, соединенные связью *Обучает*. Действительно ли *Обучает* представляет связь типа "многие к одному" в направлении от *Учебные курсы* к *Преподаватели* Ответ зависит от корпоративной политики и традиций конкретного учебного заведения. Вполне возможно, что за каждым курсом закреплен в точности один преподаватель. Даже в том случае, если несколько преподавателей ведут курс совместно, не исключено, что в соответствии с внутренними требованиями организации в базе данных должен быть указан только один из преподавателей, "ответственный" за курс. В любом из подобных случаев мы имеем право утверждать, что связь *Обучает,*

соединяющая множества сущностей *Учебные курсы* и *Преподаватели,* должна относиться к типу "многие к одному" (если рассматривать ее в указанном направлении).

С другой стороны, в учебном заведении может быть принят такой порядок, когда несколько преподавателей способны заменять друг друга при чтении определенного курса без каких бы то ни было ограничений. Либо изначальный смысл связи *Teaches* заключается не в том, чтобы отобразить зависимость между курсом и преподавателем, ведущим его в данный момент, а зафиксировать хронологию ведения курса различными преподавателями или просто описать множество преподавателей, готовых к чтению этого курса (наименование связи *Обучает* само по себе не раскрывает ее истинного смысла). В любом из названных случаев уместно считать, что связь *Обучает* должна относиться к типу “многие ко многим»

Отсутствие избыточности

Мы обязаны тщательно следить за тем, чтобы объекты структуры не повторяли друг друга. Пусть, например, сущности *Книги* и *Издательства* соединены связью *Публикуют*. Мы можем также снабдить множество сущностей *Книги* атрибутом *Издательство*. Хотя такое решение на первый взгляд вполне приемлемо, оно таит в себе ряд опасностей.

1. Повторение описания издательства книги приводит к дополнительному расходу памяти при хранении атрибута в базе данных.

2. При изменении названия издательства на новое, определяемое связью *Публикуют* в одной строке но забыть исправить в другой. Конечно, можно утверждать, что подобные ошибки вряд ли вероятны, но на практике они, к несчастью, случаются, и, пытаясь описать некий объект несколькими различными способами, мы заранее ставим себя, в невыгодное положение.

Простота

Включайте в проект только те структурные элементы, без которых совершенно нельзя обойтись.

Выбор подходящих связей

Множества сущностей могут быть соединены посредством связей разными способами. Однако включение в проект первых попавшихся связей — это не самое хорошее решение. Во-первых, оно способно привести к избыточности схемы, когда одни и те же соотношения множеств сущностей выражаются несколькими цепочками соединений.

Во-вторых, реализация подобной модели потребует дополнительного расхода дискового пространства, а задача внесения изменений в схему базы данных существенно усложнится, поскольку исправление одной связи повлечет необходимость модификации многих других.

Чтобы проиллюстрировать проблемы и описать средства их разрешения, мы воспользуемся двумя примерами: в первом несколько связей представляют одну

и ту же информацию, а во втором одна связь может быть получена на основании нескольких других.

Использование элементов адекватных типов

Иногда при выборе типа элемента модели, используемого для отображения реального объекта, возможно наличие нескольких альтернатив дальнейших действий. Во многих случаях приходится поразмышлять, чему отдать предпочтение — атрибутам или сочетаниям множеств сущностей и связей. Вообще говоря, атрибут реализовать гораздо проще, нежели множество сущностей или связь. Однако крайности всегда вредны — необоснованное повсеместное применение атрибутов способно привести к неприятностям.

Пример 2.3. Обратимся к рис. 2.2 и рассмотрим одну частную проблему. Насколько здравым было наше решение о представлении объектов Издательства отдельным множеством сущностей *Издательства?* Может быть, следовало передать атрибуты "*название издательства*" и "*адрес издательств*а" множеству сущностей *Книги* и исключить *Издательства* вовсе? Если поступить так, как мы сказали, придется повторять адрес издательства в наборе данных о каждой книге. Это еще один пример избыточности, подобный тем, которые упоминались в двух предыдущих разделах. Помимо издержек, связанных с избыточностью данных, мы столкнемся со следующей опасностью: если в базе данных будут отсутствовать сведения о каких бы то ни было книгах, выпущенных определенным издательством, информация об этом издательстве как таковом окажется утраченной.

С другой стороны, если потребность в хранении адреса издательства не возникает, никакого вреда не принесет то, что наименование издательства в качестве атрибута будет поставлено в соответствие каждой книге. Теперь об избыточности из-за многократного повторения адреса издательства речь не идет, а то, что наименование издательства, такое как, скажем, "Бином", указывается для каждой книги, изданного этим издательством, не может рассматриваться как излишняя информация, поскольку авторские права издательства должны быть отражены так или иначе.

Теперь попробуем абстрагировать выводы, к которым мы пришли, чтобы более строго описать обстоятельства, позволяющие предпочесть атрибут множеству сущностей.

3. Моделирование ограничений

До сих пор мы говорили о том, как моделировать "срез" реального мира средствами множеств сущностей, атрибутов и связей. Однако существуют некоторые другие аспекты действительности, для адекватного представления которых возможностей инструментов моделирования, изученных нами, уже не достаточно. Дополнительная информация об объектах окружающего мира зачастую бывает облечена в форму определенных *ограничений* (*constraints*),

которые никак не связаны со структурными и "типовыми" ограничениями, сопутствующими описанию конкретных множеств сущностей, атрибутов и связей.

Классификация ограничений

Следующий перечень представляет классификацию наиболее распространенных ограничений.

- ***Ключ*** (*key*) — атрибут или подмножество атрибутов, уникальным образом определяющие некоторую сущность в составе множества сущностей. Никакие две сущности в пределах множества сущностей не могут обладать одинаковыми комбинациями значений атрибутов, составляющих ключ. Впрочем, совпадение отдельных, но не всех одновременно, значений ключевых атрибутов любых двух сущностей вполне допустимо.
- ***Ограничение уникальности*** (*single-value constraint*): определенное значение э некотором контексте должно быть уникальным. Ключи — это основной "поставщик" ограничений уникальности, поскольку наличие ключа предполагает, что каждая сущность во множестве сущностей обладает уникальными значениями атрибутов, в совокупности образующих ключ. Есть и другие источники, "порождающие" ограничения уникальности, — например, связи типа "многие к одному".
- ***Ограничение ссылочной целостности*** (*referential integrity constraint*): некоторое значение, на которое ссылается другой объект, должно существовать в базе данных. Ограничение ссылочной целостности аналогично запрету на использование в обычных программах "висящих" указателей или иных элементов, ссылающихся на нечто отсутствующее.
- ***Ограничение домена*** (*domain constraint*): значение атрибута должно выбираться из некоего конечного множества значений или принадлежать определенному диапазону изменения.
- ***Ограничение общего вида*** (*general constraint*): произвольное требование, которое должно быть зафиксировано в базе данных, — например, такое как "каждой сущности *Книги* должно быть поставлено в соответствие не более десяти сущностей *Автор*.

Существует несколько причин, обусловливающих важность механизма задания ограничений в рамках модели базы данных. Ограничения дают возможность выразить некоторые особые свойства моделируемых сущностей реального мира. Так, например, ключи позволяют однозначно распознавать элементы множества однородных сущностей. Если, скажем, мы знаем, что атрибут *Название* служит ключом для множества сущностей *Издательства*, тогда при обращении к сущности *Издательство* по ее названию обеспечиваются гарантии выбора уникального элемента множества. Помимо того, при сохранении уникального значения экономятся время и дисковое пространство; отдельное значение занести в базу данных проще, нежели множество значений — даже в том случае, если множество состоит из единственного члена. Применение ключей и ограничений ссылочной целостности также способствует оптимизации структур данных, что приводит к увеличению скорости доступа к информации.

Ключи и ER-моделирование

Говоря более строго, *ключ* (key) для множества сущностей *E* — это множество *K,* состоящее из одного или более атрибутов множества E, такое, что при выборе из *E* любых двух различных сущностей e_1 и e_2, последние не могут обладать одинаковыми значениями атрибутов, относящихся к множеству *K.* Если *K* состоит из нескольких элементов, допустимо равенство отдельных, но не всех, атрибутов сущностей e_1 и e_2. Надлежит запомнить важные положения, перечисленные ниже.

- Каждое множество сущностей должно обладать определенным ключом.
- Ключ может состоять более чем из одного атрибута (соответствующая иллюстрация приведена в примере 1.1.
- Для каждого множества сущностей допускается наличие нескольких ключей. Целесообразно, однако, выбрать один их них в качестве ***"первичного ключа"*** (*primary key*) и далее обращаться с множеством таким образом, словно оно обладает единственным ключом.
- Если некоторое множество сущностей вовлечено в иерархию связей ***isa,*** следует обеспечить, чтобы корневое множество сущностей обладало всеми атрибутами, необходимыми для формирования ключа, и ключ для каждой сущности мог быть найден на основе ее компонентов, относящихся к корневому множеству сущностей, — независимо от того, какое число множеств, участвующих в иерархии, располагает соответствующими компонентами.

Пример 3.1. Рассмотрим множество сущностей *Книги*. На первый взгляд, на роль ключа подходит атрибут *Название*. Однако нетрудно привести целый ряд примеров названий, которые употреблялись для обозначения двух и более различных книг, скажем «Системы баз данных». Поэтому выбор атрибута *Название* в качестве самодостаточного ключа — это, очевидно, не самое хорошее решение. Если все же поступить именно так, в дальнейшем мы не сможем различить в базе данных те книги, которые получили одно и то же название.

Более уместно создать ключ на основе двух атрибутов — *Название* и *Год*. Разумеется, вероятность выхода в одном и том же году двух книг с одинаковым названием (в базе данных нельзя будет представить сведения об одном и другом одновременно) все еще существует, но она, надо признать, довольно мала.

Решение о выборе ключей для двух других основных множеств сущностей, представляющих рассматриваемую нами "книжную" базу данных, должно быть не менее взвешенным. Функции ключа для множества сущностей *Издательства* имеет смысл возложить на атрибут *Название*, поскольку двух издательств с одинаковым названием определенно быть не может. Однако столь же категорично утверждать, что сущность *Автор* однозначно определяется атрибутом *Фамилия*, мы бы не стали, поскольку различить людей только по имени-фамилии в общем случае нельзя. Впрочем, авторы довольно часто выбирают для себя броские "сценические" псевдонимы, поэтому

атрибут *Фамилия* может рассматриваться как претендент на роль ключа множества сущностей *Авторы*. Более универсальным, однако, могло бы оказаться решение, предполагающее создание ключа на основе пары атрибутов, *name* и *address* ("адрес"), — если, конечно, вы не столкнетесь с тем редким случаем, когда два автора с одинаковым именем проживают по одному и тому же адресу.

Ограничения как часть схемы базы данных

При изучении содержимого некоторой базы данных легко впасть в заблуждение и принять решение о выборе ключевых атрибутов лишь на том, довольно зыбком, основании, что в данный момент времени никакие из сущностей рассматриваемого множества не обладают одинаковыми значениями этих атрибутов (скажем, в базу данных до сих пор не заносились сведения о кинофильмах с одинаковыми названиями). Таким образом, для множества сущностей *Книги* ("кинофильмы") ключ может быть ошибочно сформирован, например, из единственного атрибута *title* ("название"). К чему это способно привести, подробно пояснять, вероятно, не нужно: если база данных проектируется с учетом того, что *title* является уникальным ключом множества сущностей *Книги,* в дальнейшем в нее нельзя будет внести информацию о нескольких одноименных кинофильмах (таких как "King Kong").

Таким образом, ограничения, касающиеся ключей, — впрочем, и ограничения всех других категорий — в процессе реализации базы данных становятся частью ее схемы, наряду со всеми остальными структурными элементами, такими как множества сущностей и связи. После того как ограничение объявлено, любые операции по пополнению или изменению содержимого базы данных, нарушающие условие этого ограничения, будут запрещены.

Хотя частный экземпляр базы данных может заведомо удовлетворять некоторым ограничениям, "подлинными" следует считать только те из них, которые определены проектировщиком базы данных и справедливы для всех ее экземпляров, верно отображающих особенности моделируемых сущностей.

Пример 3.2. Может показаться, что задача выбора подходящих ключей, гарантирующих уникальность представляемых данных, достаточно сложна. В действительности обычно все гораздо проще. В реальных ситуациях, подлежащих моделированию, при выборе ключевых атрибутов множеств сущностей зачастую применяются непрямолинейные подходы. Например, во многих компаниях принято присваивать всем служащим уникальный идентификационный код, и одна из целей подобных мер состоит в том, чтобы облегчить введение информации о служащих в корпоративные базы данных и гарантировать возможность различения записей даже в тех случаях, когда несколько служащих имеют одинаковые имена и фамилии. Таким образом, в

качестве ключа множества сущностей, описывающего служащих, используется атрибут идентификационного кода. Для студента – это номер зачетной книжки.

В России является нормой, когда каждый житель обладает также собственным индивидуальным номером налогоплательщика (ИНН) или номером полиса медицинского страхования. Если в базе данных предусмотрен соответствующий атрибут для отображения номера полиса медицинского страхования служащих, он также вполне пригоден для использования в качестве ключевого. Заметим, что в наличии нескольких альтернативных вариантов ключей для одного и того же множества сущностей — например, самостоятельных ключевых атрибутов идентификационного кода и номера полиса медицинского страхования служащих — нет ничего предосудительного.

Практика создания специальных атрибутов, единственным назначением которых является поддержка ключевых признаков сущности во множестве однородных сущностей, достаточно распространена. Аналогично идентификаторам служащих, во всех университетах каждому студенту также присваивают уникальный идентификационный код - номер зачетной книжки. В базах данных Государственной автомобильной инспекции в качестве ключевых атрибутов множеств сущностей, представляющих сведения о водителях и транспортных средствах, используются, соответственно, номера водительских удостоверений и регистрационные номера транспортных средств. Любой читатель при желании сможет привести и другие примеры определения специальных атрибутов, предназначенных для использования в качестве ключевых.

Представление ключей в ER-модели

Используемая нами система обозначений предусматривает, что атрибуты множеств сущностей, объявленные как ключевые, на ER-диаграмме выделяются ***подчеркиванием***. Вернемся к рисунку 1.2, который отображает структуру базы данных, содержащей сведения о книгах *(Книги),* издательствах *(Издательства)* и авторах *(Авторы).* Ключевые атрибуты названных множеств сущностей подчеркнуты. В качестве ключевых атрибутов для множеств *Авторы* и *Издательства* выбраны атрибуты *Имя* и *Название*.

Ключ для множества сущностей *Книги* сформирован на основе совокупности атрибутов *Название* и *Год*. Атрибуты, наименования которых *подчеркнуты,* являются членами ключа. Для описания ситуации, когда множество сущностей снабжено несколькими альтернативными ключами, специальных обозначений, однако, не существует; мы подчеркиваем только ***первичные ключи*** (primary keys).

Ограничения уникальности

Важным свойством некоторого объекта базы данных иногда является то, что он может описываться не более чем одним значением. Так, например, конкретной сущности *Книга* соответствуют строго определенные значения

атрибутов "Название", "Год", "Объем" и "Тип", а "опубликовала" книгу только одно издательство.

Существует несколько ситуаций, в которых ***ограничения уникальности*** (*single-value constraints*) проявляют себя в рамках ER-модели.

1. Некоторый атрибут множества сущностей обладает единственным значением. Иногда допускается отсутствие значения атрибута для отдельных сущностей множества, и в подобных случаях приходится "изобретать" какое-то "нулевое" значение, используемое по умолчанию. Для примера можно представить ситуацию, когда для некоторых книг, сведения о которых занесены в базу данных, информация об их объеме неизвестна. Значением атрибута *Объем* в таком случае могло бы быть, скажем, —1. С другой стороны, допускать отсутствие значений ключевых атрибутов, таких как *Название* и *Год*, нельзя. Требование о том, что некоторый атрибут не может иметь неопределенных значений вида *null,* в ER-модели не имеет специальных средств представления. Если необходимо описать подобное ограничение, можно снабдить атрибут дополнительным текстовым пояснением.

2. Связь *R* типа "многие к одному", соединяющая множества сущностей *E* и *F* в направлении от *E* к *F,* подразумевает наличие ограничения уникальности. Другими словами, для каждой сущности *e* множества *E* имеется не более одной связанной с ней сущности f множества *F.* Или, в общем случае, если *R* представляет собой многостороннюю связь, тогда каждая из стрелок, "выходящая" из *R,* выражает ограничение уникальности. В частности, если существует стрелка, соединяющая связь *R c* множеством сущностей *E,* в составе *E* имеется не более одной сущности, ассоциированной с определенным набором сущностей из всех остальных множеств, охватываемых связью *R.*

Ограничения ссылочной целостности

Если ограничения уникальности (*single-value constraints*) декларируют возможность существования не более одного значения, выступающего в определенной роли, ***ограничения ссылочной целостности*** (*referential integrity constraints*) подразумевают наличие в точности одного такого значения. Требование о том, чтобы атрибут обладал единственным возможным значением, отличным от *null,* можно рассматривать как разновидность ограничений ссылочной целостности, но чаще последние используются для описания свойств *связей* между множествами сущностей.

Рассмотрим связь *Публикуют*, соединяющую множества сущностей *Книги* и *Издательства* так, как показано на рис. 1.2. Связь *Публикуют* относится к типу "многие к одному" и предполагает только то, что ни одна из книг не может публиковаться нескольким издательствами одновременно. Связь — в том виде, как она задана — отнюдь *не* подразумевает, что кинофильмом определенно *должна* владеть какая-либо студия, а если это даже и так, студия вовсе *не* обязана быть представлена во множестве сущностей *Издательства.*

Ограничение ссылочной целостности в контексте связи *Публикуют* предусматривает, что для каждой сущности "книга" в базе данных должна присутствовать соответствующая сущность "издательство", на которую "книга" ссылается. Существует несколько способов поддержки условий ссылочной целостности. Мы рассмотрим их ниже на примере связи *Публикуют.*

1. Если база данных пополняется новой сущностью "книга", обязана существовать (или должна быть добавлена) и соответствующая сущность " издательство ". Помимо того, если значение экземпляра связи *Публикуют* (ссылка на множество сущностей *Издательства)* для конкретного "книга" изменяется, необходимо гарантировать, что новое значение ссылки также является сущностью множества *Издательства.*

2. Следует запретить удаление сущностей множества *Издательства,* адресуемых сущностями множества *Книги.* Другими словами, некоторая сущность " издательство " не может быть удалена до тех пор, пока имеются сущности "книга", которые на нее ссылаются.

3. Если сущность множества *Издательства* подлежит *принудительному* удалению, должны быть удалены также все сущности множества *Книги,* ссылающиеся на нее.

Аспекты практической реализации ограничений ссылочной целостности относятся к компетенции определенной СУБД или конкретного проекта базы данных, поэтому здесь мы не будем останавливаться на них подробно.

Ссылочная целостность и ER-диаграммы

Правила применения стрелок в ER-диаграммах могут быть расширены с целью обеспечения возможности отображения ограничений ссылочной целостности связей в одном или нескольких направлениях. Предположим, что *R* — это связь, соединяющая множества сущностей *E* и *F* в направлении от *E* к *F*. Тогда *полукруглая* стрелка, указывающая на *F,* не только свидетельствует о том, что связь, соединяющая множества *E* и *F,* относится к типу "многие к одному" или "один к одному", но и отображает требование обязательного наличия в *F* сущности, на которую ссылается определенная сущность из *E.* Те же соображения применимы и к случаю, когда *R* соединяет между собой более двух множеств сущностей.

Пример 3.4. На рис. 3.2 обозначены несколько ограничений ссылочной целостности, заданных для связей *Публикуют* и *Возглавляет*, которые соединяют множества сущностей *Книги* ("книги"), *Издательства* и *Президенты*. Полукруглая стрелка, соединяющая связь *Публикуют* с множеством сущностей *Издательства,* выражает ограничение ссылочной целостности, которое подразумевает, что каждую книгу должно публиковать одно издательство и соответствующая сущность издательство обязана присутствовать в составе множества *Издательства.*

Рис. 3.2. ER-диаграмма с обозначениями ограничений ссылочной целостности

Аналогично, полукруглая стрелка, указывающая на множество *Издательства* со стороны связи *Управляет,* обозначает ограничение ссылочной целостности, которое заключается в том, что каждый президент возглавляет одну студию и соответствующая сущность "киностудия" должна наличествовать во множестве *Издательства.*

Обратите внимание, связь *Управляет* по-прежнему соединена с множеством сущностей *Директор* обычной (а не полукруглой) стрелкой, поскольку она отображает вполне естественную закономерность взаимоотношений сущностей "*Директор*" и "*Издательства*". Если студия прекращает свое существование, ее президент более не может занимать свой пост, и поэтому резонно ожидать, что соответствующая сущность "президент" должна быть удалена из множества *Директор;* здесь проявляет себя ограничение ссылочной целостности. С другой стороны, удаление сущности "президент", как подсказывает здравый смысл, не должно оказывать влияния на содержимое множества *Издательства,* поскольку в какие-то периоды времени студия может обходиться без руководителя. Поэтому связь *Управляет* и множество сущностей *Директор* соединены обычной стрелкой: каждая студия может возглавляться не более чем одним президентом, а в каких-то случаях президента может не быть вовсе.

4. Слабые множества сущностей

Изредка обстоятельства складываются так, что ключ для некоторого множества сущностей (его называют *слабым — weak entity set*) приходится формировать на основе атрибутов, которые полностью или частично принадлежат другому множеству сущностей.

Примеры использования слабых множеств сущностей

Существуют два основных источника, порождающих необходимость использования слабых множеств сущностей. Во-первых, иногда множества сущностей участвуют в иерархии, построенной на принципах, не имеющих отношения к связям ***isa*** Если сущности множества *E* представляют собой единицы более "крупных" сущностей множества F, вполне вероятна ситуация, когда атрибуты сущностей *E,* претендующие на роль ключевых, не будут обладать свойством уникальности до тех пор, пока во внимание не приняты атрибуты сущностей *F,* которым "подчиняются" сущности *E.* Проблема иллюстрируется несколькими примерами.

Пример 4.1. Кафедра может состоять из нескольких лабораторий. Пусть объединениям (назовем их *Лаборатории)* в рамках кафедры даны условные названия, скажем, "Лаборатория-1", "Лаборатория-2" и т.д. Но в другой кафедре для нумерации лабораторий может быть принята совершенно такая же схема, так что атрибут *Номер* уже нельзя считать ключевым. Чтобы обеспечить уникальность названия лаборатории среди подразделений всех кафедр, необходимо, наряду с его номером, учесть и наименование *(name)* кафедры, которой эта лаборатория принадлежит. Ситуация смоделирована на рис. 4.1. Ключом для слабого множества сущностей *Crews* служит совокупность двух атрибутов — собственного атрибута *Номер* множества *Лаборатории* и атрибута *Название* уникальной сущности-Кафедры, с которой сущность-"Лаборатории" соотносится посредством связи *Находятся на*.

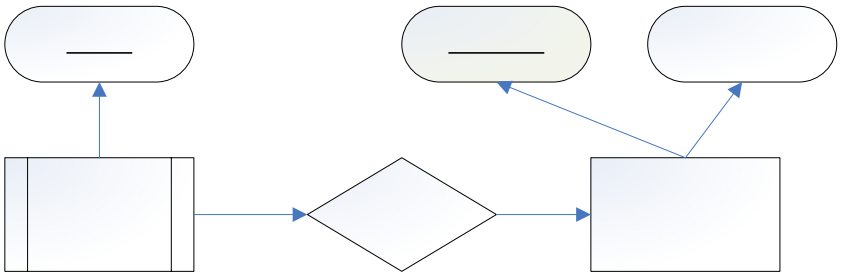


Рисунок 4.1. Так отображаются слабые множества

5. Реляционные модели данных

Метод описания данных E/R, вполне подходят для описания структуры данных, однако современные реализации БД почти всегда базируются на другой модели — ***реляционной***. Ее ценность заключается в том, что она основана на единственном понятии моделирования данных — "отношении", двумерной таблице, в которой собраны данные. Реляционная модель поддерживает язык программирования очень высокого уровня — SQL, позволяющий писать простые и мощные программы, которые манипулируют данными, хранящимися в отношениях.

Тем не менее иногда бывает легче проектировать БД с помощью моделей типа ER, описанных в предыдущем разделе. Поэтому необходимо знать, как перевести проекты из E/R в отношения. Реляционная модель имеет собственную теорию, которая часто называется нормализацией отношений и основана на "функциональных зависимостях", Теория нормализации часто облегчает выбор отношений, представляющих конкретный проект БД.

Основы реляционной модели

Реляционная модель обеспечивает единственный способ представления данных в виде двумерной таблицы, называемой ***отношением***. Пример отношения приведен на рис. 5.1.

Название	Год	Объем	Тип
Системы баз данных	2003	1071	Учебник
Visual Basic.NET	2003	450	Учебник
Ассемблер	2004	635	Учебник
Незапертая дверь	2003	332	Детектив

Рис. 5.1. Отношение *Книги*

Это отношение имеет имя *Книги* и имеет только атрибуты *Название, Год, Объем и Тип*.

Атрибуты

В верхней части рис. 5.1 расположены ***атрибуты*** Название, Год, Объем и Тип. Атрибуты отношения служат именами его столбцов. Обычно атрибут отражает смысл того, что записано в лежащем под ним столбце. Например, в столбце с атрибутом Объем указано количество страниц книги.

Схемы

Имя отношения и множество его атрибутов образуют ***схему*** этого отношения. Схема обозначается именем отношения, за которым следует заключенный в круглые скобки список его атрибутов. Схема отношения Книги на рис. 5.1:

Книги (Название, Год, Объем, Тип)

В реляционной модели проект состоит из одной или нескольких схем отношений. Множество схем отношения в проекте называется ***схемой реляционной БД*** или просто ***схемой БД***.

Кортежи

Строки отношения, отличные от заглавной строки, состоящей из атрибутов, называются ***кортежами***. Кортеж содержит один *компонент* для каждого атрибута отношения. Например:

(Системы баз данных, 2003, 1071, Учебник)

является кортежем рис. 5. При появлении кортежа отдельно от отношения атрибуты отсутствуют, поэтому необходимо каким-то образом указать отношение, к которому он принадлежит. Во всех случаях мы будем придерживаться порядка, при котором атрибуты были перечислены в схеме отношения.

Отношения - это множества кортежей, причем один кортеж не может дважды входить в данное отношение. Поэтому, если отношение представляется как объект, нужна гарантия, что у отношения достаточно атрибутов для того, что два каких-либо объекта не имели одинаковых значений всех атрибутов данного отношения.

Домены

Как того требует реляционная модель, каждый компонент каждого кортежа должен быть атомарным, т.е. компонентом элементарного типа, например целым числом или строкой. Недопустимо, чтобы значение было структурой записи, множеством, списком, набором или имело любой иной тип, который можно разит на более мелкие компоненты

Кроме того, предполагается, что с каждым атрибутом отношения связана какая-то ***область значений****, т.е.* отдельный элементарный тип. Каждый компонент любого кортежа отношения должен иметь значение, принадлежащее области соответствующего столбца. Например, первым компонентом каждого из кортеже отношения Книги должна быть строка, вторым и третьим — целое число, а значением четвертого компонента — одна из констант Учебник или Детектив.

Экземпляры отношения

Отношение, касающееся книг, не статично, со временем оно изменяется. Предполагается, что изменяются и кортежи отношения: в БД вводятся новые кортежи книг, существующие кортежи изменяются при получении новой информации о книге и, бывает, удаляются кортежи для тех книг, которые по какой-то причине исключаются из БД.

Схема отношения изменяется реже. Однако в некоторых ситуациях необходимо добавлять или удалять атрибуты. Схема при этом изменяется, а это в некоторых коммерческих системах БД может стоить очень дорого, так как для добавления или удаления компонента приходится переписывать иногда миллионы кортежей. При добавлении атрибута иногда трудно, а подчас даже невозможно найти правильные значения новых компонентов кортежей

Множество кортежей для заданного отношения будем называть ***экземпляром*** этого отношения. Например, четыре кортежа на рис. 5 составляют экземпляр отношения *Книги*.

Схемы и экземпляры

Не забывайте о важных различиях между схемой отношения и его экземпляром. Схема — это имя и атрибуты отношения, она относительно неизменна.

Экземпляр — это множество кортежей для данного отношения, он может часто изменяться.

6. Операции в реляционной модели

Посмотрим на БД с точки зрения пользователя. Обычно главное для пользователя — ***запросы*** к БД, т.е. написание программ, отвечающих на вопросы о текущем состоянии БД.

Реляционная модель содержит конкретное множество стандартных операций над данными. Поэтому при анализе работы с БД мы сосредоточимся

на реляционной модели и ее операциях, которые можно выразить в алгебре, называемой реляционной алгеброй.

Позже мы рассмотрим языки и средства, которые в настоящее время предлагают пользователю коммерческие системы БД. Абстрактные операторы запросов реализуются в виде операций языка запросов SQL.

Алгебра реляционных операций

Приступая к изучению операций на отношениях, нужно познакомиться со специальной ***алгеброй — реляционной***. Она состоит из простых, но мощных методов конструирования новых отношений из уже имеющихся. ***Выражения*** реляционной алгебры начинаются с отношений в качестве операндов. Отношения представляются их именами {например, *R* или Книги) или явно, в виде списка их кортежей. Затем можно последовательно строить сложные выражения, применяя любые описанные ниже операторы к отношениям или к более простым выражениям реляционной алгебры. ***Запрос-*** это выражение реляционной алгебры. Таким образом, реляционная алгебра — первый конкретный пример языка запросов.

Операции реляционной алгебры делятся на четыре распространенных класса

1. Обычные теоретико-множественные операции: объединение, пересечение и разность, примененные к отношениям.
2. Операции удаления частей отношения: "отбор", удаляющий некоторые строки (кортежи), и "проекция", удаляющая некоторые столбцы.
3. Операции комбинирования кортежей двух отношений, включая "декартово произведение" (образование пар кортежей двух отношений всеми возможными способами) и множество видов "соединения" – выборочного образования пар кортежей двух отношений.
4. Операция "переименования", которая не влияет на кортежи отношения, но изменяет реляционную схему, т.е. имена атрибутов и/или имя самого отношения.

Для выполнения всех возможных вычислений на отношениях этих операций недостаточно; их действие весьма ограничено. Однако они позволяют сделать многое из того, что необходимо делать с БД и составляют большую часть стандартного реляционного языка запросов SQL

Теоретико-множественные операции на отношениях

Наиболее распространенные операции множеств — объединение, пересечение и разность. Для произвольных множеств *R* и *S* они определяются следующим

образом.

- ***R*** **U** ***S****, объединение R* и *S*— это множество элементов, входящих или в *R,* или в S, или в оба множества. Элемент входит в объединение только один раз, даже если он входит и в R, и в *S.*
- ***R*** ∩ ***S,*** *пересечение R* и *S* — это множество элементов, входящих и в R, и в *S.*

- ***R - S****, разность R* и *S*- это множество элементов, которые входят в *R,* но не входят в *S.* Заметим, что *R - S* отличается от *S - R;* второе выражение обозначает множество элементов, входящих в S, но не входящих в R.

При применении этих операций к отношениям необходимо соблюдать ряд условий.

1.R и S должны иметь схемы с идентичными множествами атрибутов.

2.Перед вычислением теоретико-множественного объединения, пересечения или разности множеств кортежей столбцы *R* и *S* должны быть расположены так, чтобы порядок атрибутов был одинаков в обоих отношениях.

Иногда нужно вычислить объединение, пересечение или разность отношений, имеющих одинаковое число атрибутов с различными именами. В таком случае применяется оператор переименования для изменения схемы одного или обоих отношений.

Название	**Год**	**Объем**	**Тип**
Системы баз данных	2003	1071	Учебник
Visual Basic.NET	2003	450	Учебник
Ассемблер	2004	635	Учебник
Незапертая дверь	2003	332	Детектив

Рис.6.1. Отношение Книги

Проекция

Оператор *проекции* применяется к отношению R для получения нового отношения, содержащего только некоторые атрибуты *R.* Значение выражения π (R) отношение, содержащее только столбцы A_1, A_2, ..., A_n отношения R. Схема результирующего значения - это множество атрибутов $\{A_1, A_2, ..., A_n\}$, которое обычно показывается в порядке перечисленных атрибутов.

Пример 6.1. Рассмотрим отношение Книги со схемой, Книги (*Название, Год*, *Объем, Тип*). Экземпляр этого отношения показан на рис. 6.1. Его можно проецировать на три первых атрибута с помощью выражения

π Название, Год, Объем (Книги)

В результате получается отношение

Название	**Год**	**Объем**
Системы баз данных	2003	1071
Visual Basic.NET	2003	450
Ассемблер	2004	635
Незапертая дверь	2003	332

Рис. 6.2. Результат проекции

Селекция

Оператор ***селекции*** или ***отбора,*** примененный к отношению *R,* дает новое отношение с подмножеством кортежей *R.* Кортежи нового отношения удовлетворяют некоторому условию ***C,*** включающему в себя атрибуты *R.* Операция отбора обозначается $\sigma_c(R)$. Схема результирующего отношения совпадает со схемой *R,* и атрибуты в ней обычно показаны в том же порядке, в каком они входят в *R.*

С—выражение условного типа. Такие выражения применяются в обычных языках программирования. Например, в языках C или Pascal условное выражение следует за ключевым словом if. Единственное различие состоит в том, что операнды в C - это константы или атрибуты *R.* Оператор *C* применяется к каждому кортежу *t* отношения *R,* при этом каждый атрибут *A,* входящий в условие C, заменяется компонентом кортежа t для атрибута *A.* Если после таких подстановок вместо каждого атрибута C это условие истинно, то кортеж *t* включается в результат $\boldsymbol{\sigma_c(R)}$**,** в противном случае он в результат не входит.

Пример 6.2.: Найти книги объемом больше 500 страниц.

Запрос выглядит следующим образом

$\boldsymbol{\sigma_{\text{объем} > 500}(R)}$

Результат – два кортежа:

Название	Год	Объем
Системы баз данных	2003	1071
Ассемблер	2004	635

Рис. 6.3. Результат селекции – выбора

Декартово произведение

Декартово произведение (Cartesian product) (или просто *произведение)* двух множеств *R* и *S*- представляет собой множество пар, причем первым элементом каждой пары является элемент множества R, а вторым — элемент множества *S.* Произведение обозначается *R* x S. Если R и *S*-отношения, их произведение, по существу, остается таким же. Но поскольку членами *R* и *S* являются кортежи, состоящие обычно из нескольких компонентов, в результате соединения кортежа из *R* с кортежем из S получается более длинный кортеж, содержащий по одному компоненту для каждого компонента кортежей составляющих данную пару. При этом компоненты из *R* предшествуют компонентам из *S.*

Реляционная схема для результирующего отношения - это объединение схем для *R* и *S.* Если же *R* и *S* имеют какие-то общие атрибуты, необходимо ввести новое имя по крайней мере для одного атрибута из каждой пары идентичных атрибутов. Чтобы уточнить, откуда взят атрибут

A, входящий в схемы *R* и S, мы пишем *R.A* для атрибута из R и *S.A* для атрибута из *S*

Пример 6.3. Заданы два отношения R и S

Отношение R*:*

A	*B*
1	*2*
3	*4*

Отношение S*:*

B	*C*	*D*
2	*5*	*6*
4	*7*	*8*
9	*10*	*11*

R x S

A	*R.B*	*S.B*	*C*	*D*
1	*2*	*2*	5	6
1	*2*	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

Рис. 6.4. Два отношения и их декартово произведение

Естественное соединение

Полезность операции декартова произведения несомненна, но гораздо чаще возникает потребность в *соединении* (*join*) двух отношений, т.е. формировании пар таких их кортежей, которые удовлетворяют определенному критерию. Простейшая разновидность операции соединения двух отношений *R* и S, которую называют *естественным соединением* (*natural join*) и обозначают как

$$R >< S,$$

предусматривает включение в итоговое отношение тех кортежей из *R* и S, которые совпадают в атрибутах, общих для схем *R* и S. Чтобы быть более точными, обозначим как *A, B, C* те атрибуты, которые определены одновременно в схемах отношений R и S. Тогда кортежи r из *R* и *s* из S допускают соединение, если и только если r и s совпадают в каждом из атрибутов *A, B, C,*.

Если кортежи r и *s* успешно соединены, результат представляет собой кортеж, называемый *соединенным кортежем* (*joined tuple*) и содержащий по одному компоненту для каждого из атрибутов схемы, полученной в результате объединения схем *R* и S. Соединенный кортеж совпадает с кортежем r в каждом из атрибутов схемы *R* и с кортежем s в каждом из атрибутов схемы S.

Поскольку r и s успешно соединены, итоговый кортеж совпадает с обоими исходными кортежами во всех их общих атрибутах.

Пример 6.4. Результатом операции естественного соединения отношений *R* и S, изображенных на рис. 6.4, окажется следующий экземпляр отношения:

A	*B*	C	*D*
1	2	5	*6*
3	4	7	*8*

Единственный атрибут, общий для отношений *R* и S, — это *B*. Поэтому для успешного соединения кортежей достаточно их совпадения в атрибуте *B*. Соединенный кортеж обладает такими атрибутами: *A* (из схемы R), *B* (из *R* или S), *C* (из S) и *D* (также из S).

Первый кортеж отношения *R* успешно сочетается только с первым кортежем отношения S; кортежи совпадают в единственном общем атрибуте *B*. В результате мы получаем первый кортеж итогового отношения: (1, 2, 5, 6). Второй кортеж *R* образует пару только со вторым кортежем S, и эта операция дает в итоге кортеж (3, 4, 7, 8). Обратите внимание, что третий кортеж S не совпадает в атрибуте B ни с одним кортежем *R* и поэтому в расчет не принимается.

Тета-соединение

Тета-соединение отношений R и S, основанное на условии C, обозначается как $R ><_c S$. Результат этой операции получается следующим образом.

1. Берется декартово произведение *R* и *S*.
2. Из полученного произведения выбираются только те кортежи, которые удовлетворяют условию C

Как и при операции произведения, схема результата — это объединение схем *R* и S, в котором префиксы "R." и *"S."* используются, чтобы указать, из какой схемы взят данный атрибут.

Пример 6.4.. Рассмотрим операцию $U><_{A<D} V$, где *U* и *V*- отношения из рис. 6.5. Необходимо рассмотреть все девять пар кортежей и проверить, действительно ли компонент *A* кортежа из *V* меньше компонента *D* кортежа из *V*. Компонент *A* первого кортежа из *U* равен 1 и успешно соединяется с каждым кортежем из *V*. Однако второй и третий кортежи из *U с* компонентами V , равными 6 и 9, успешно соединяются только с последним кортежем *V*. Значит, результат содержит только пять кортежей, построенных с помощью пяти успешных соединений в пары. Это отношение показано на рис. 6.5.

A	B	C
1	2	3
6	7	8
9	7	8

Отношение U

B	C	D
2	3	4
2	3	5
7	8	10

Отношение V

A	U. B	U.C	V.B	V.C	D
1	2	3	2	3	4
1	2	3	2	3	5
1	2	3	7	8	10
6	7	8	7	8	10
9	7	8	7	8	10

Рис. 6.5. Пример тета – соединения $U \mathbin{><}_{(A < D)} V$

Заметим, что схема результата содержит все шесть атрибутов. Префиксы *U* и *V* позволяют различить соответствующие им вхождения атрибутов и и С. Таким образом. тета-соединение отличается от натурального соединения, так как в последнем общие атрибуты смешаны в одной копии. Конечно, в натуральном соединении это уместно, поскольку кортежи не соединяются, если они не совпадают по общим атрибутам. В случае тета-соединения нет гарантии, что сравниваемые атрибуты в результате совпадут, так как оператором сравнения необязательно является

Комбинирование операций для построения запросов

Если бы в качестве запроса разрешалось писать только единичные операции на одном или двух отношениях, от реляционной алгебры было бы мало пользы. Однако она, как и все алгебры, позволяет строить выражения произвольной сложности путем применения операторов к заданным отношениям или к отношениям. уже полученным в результате применения операторов.

Выражения реляционной алгебры можно строить, применяя операции к более частным выражениям и при необходимости выделяя скобками группы операндов. Выражения представляются в виде дерева, чтобы их легче было прочесть, хотя они меньше подходят в качестве нотации для машинного чтения.

Пример 6.4. Рассмотрим декомпозированное отношение *Книги* из примера 1.1. Допустим, нужно узнать названия и годы выпуска книг студии Вильямс, которые имеют объем более 500 стр. Это можно вычислить следующим образом.

1. Выбрать кортежи Книги, имеющие *объем* > 500.
2. Выбрать кортежи Книги, имеющие название издательства — Вильямс.
3. Вычислить пересечение (1) и (2).

4. Построить проекцию отношения из пункта (3) на атрибуты *Название* и *Год*.

Это же выражение можно представить в обычной линейной нотации со скобками, а именно, в виде формулы

Эквивалентные выражения и оптимизация запросов

Все системы БД включают в себя системы ответов на запросы, многие из которых основаны на языках, по своей выразительности равных реляционной алгебре. Поэтому запрос пользователя может иметь множество *эквивалентных выражений* (выражений, дающих один и тот же ответ, когда в них в качестве операнд используются одни и те же отношения). Некоторые из них можно оценить намного быстрее, чем другие.

Переименование

Контроль за именами атрибутов, используемых в отношениях, которые построены с помощью одной или нескольких операций реляционной алгебры, полезно осуществлять с помощью оператора, явным образом переименовывающего эти отношения. Для переименования отношения *R* будем использовать оператор ρ_s ***(A, B, C)*(R)-** Результирующее отношение имеет в точности те же кортежи, что и *R* с именем отношения *S.* Его атрибуты получают имена ***A, B, C*** в порядке слева направо. Если нужно изменить только имя отношения, но не его атрибутов, можно писать просто ρ_s **(R).**

Зависимые и независимые операции

Некоторые операции, описанные в разделе 4.1, можно выразить в терминах других операций реляционной алгебры, например, операция пересечения в терминах разности множеств имеет вид:

$$R \cap S = R - (R - S)$$

Значит, если *R* и *S*~ отношения с одной и той же схемой, их пересечение можно вычислить следующим образом. Сначала выделяем *S* из *R,* чтобы получить отношение *T,* состоящее из кортежей, входящих в R, но не в *S, а* затем выделяем T из R. оставляя только те кортежи *R,* которые входят в *S.*

Два вида соединения тоже можно выразить в терминах других операций. Тета-соединение записываем с помощью операции декартова произведения и отбора:

$$R >< S = \sigma_c(R \times S),$$

Натуральное объединение можно выразить, начиная с произведения *RxS,* к которому затем применяется оператор отбора с условием *C* вида

$R.A_1 = S.Ai$ AND $R.A_2 = S.A_2$ AND ... AND $R.An,n, = SA_n$

где А,, А, ... *A,,* -- все атрибуты, появляющиеся в схемах *R* и *S.* И наконец, нужно построить проекцию одной копии каждого из приравненных друг другу атрибутов. Пусть *L* — список атрибутов в схеме *R,* за которым следуют

те атрибуты из схемы 5, которые не входят в схему *R*. Тогда выражение приобретет следующий вид:

$$\mathbf{R >< S = \sigma_c(R \times S)}$$

Пример базы данных:

Здесь приведем пример реляционной схемы базы данных и экземплярами отношений. База данных состоит из четырех отношений, схемы которых выглядят так:

Product(maker, model, type),
PC(model, speed, ram, hd, rd, price),
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)

Отношение Product (*продукты*) представляет информацию о производителях (*maker*), номерах моделей (*model*) и типах (*type*) различных образцов компьютерного оборудования — персональных компьютеров (*pc*), переносных компьютеров (*laptop*) и принтеров (*printer*)- Мы подразумеваем, что номера моделей продуктов различных типов, выпускаемых разными производителями, следуют одному формату; строго говоря, такое допущение далеко от действительности, и в реальных базах данных в номер модели обычно включается определенный код производителя. Отношение PC ("персональные компьютеры") для каждой модели (*model*) компьютера содержит значения быстродействия процессора (*speed*) в мегагерцах, объема оперативной памяти (*ram*) в мегабайтах, емкости жесткого диска (*hd*) в гигабайтах, скорости и типа привода съемных дисков (rd) — CD или DVD, а также цены (*price*) в долларах США. Схема отношения Laptop ("переносные компьютеры") почти аналогична схеме отношения PC, за исключением того, что вместо атрибута rd в ней предусмотрен атрибут screen, предназначенный для хранения данных о размере экрана в дюймах. Отношение Printer ("принтеры") для каждой модели (*model*) принтера содержит информацию о том, является ли принтер цветным (*color*) (*true*, если да, и *false* — в противном случае), о типе (type) технологии, поддерживаемой принтером (*'laser'* - лазерный, *'ink-jet'* - струйный, *'bubble'* — пузырьковый), а также о его цене (*price*).

Экземпляр отношения Product показан на рис. 6.7, а экземпляры трех остальных отношений — на рис. 6.8. Наименования производителей и номера моделей упрощены и унифицированы, но остальные данные вполне реальны.

Напишите выражения реляционной алгебры, способные дать ответы на перечисленные ниже запросы. Если хотите, можете применить линейную форму представления запросов в виде наборов операторов присваивания. Для иллюстрации результатов используйте информацию из экземпляров отношений рис. 6.7 и 6.8.

a) Какие модели персональных компьютеров обладают процессорами с быстродействием не ниже 1000 МГц?
b) Какие производители выпускают переносные компьютеры с жесткими дисками емкостью не ниже 1 Гбайт?

c) Найти номера моделей и цены всех продуктов (любых типов), выпускаемых производителем 'B'.
d) Найти номера моделей всех цветных лазерных принтеров.
e) Найти тех производителей, которые выпускают только переносные, но не персональные компьютеры.
f) Найти те значения емкости жесткого диска, которые характерны для двух или более персональных компьютеров.

Product

maker	model	type
A	1001	pc
A	1002	pc
A	1003	pc
A	2004	laptop
A	2005	laptop
A	2006	laptop
B	1004	pc
B	1005	pc
в	1006	pc
в	2001	laptop
в	2002	laptop
в	2003	laptop
c	1007	pc
c	1008	pc
c	2008	laptop
c	2009	laptop
c	3002	printer
c	3003	printer
c	3006	printer
D	1009	pc
D	1010	pc
D	1011	pc
D	2007	laptop
E	1012	pc
E	1013	pc
E	2010	laptop
F	3001	printer
F	3004	printer
G	3005	printer
H	3007	printer

Рис. 6.7. Экземпляр отношения Product

PC

model	speed	ram	hd	rd	price
1001	700	64	10	48xCD	799
1002	1500	128	60	12xDVD	2499

1003	866	128	20	SxDVD	1999
1004	866	64	10	12XDVD	999
1005	1000	128	20	12xDVD	1499
1006	1300	256	40	16xDVD	2119
1007	1400	128	80	12xDVD	2299
1008	700	64	30	24xCD	999
1009	1200	128	80	16xDVD	1699
1010	750	64	30	40xCD	699
1011	1100	128	60	16xDVD	1299
1012	350	64	7	48xCD	799
1013	733	256	60	12xDVD	2499

(a) Экземпляр отношения PC

LAPTOP

model	speed	ram	hd	screen	price
2001	700	64	5	12.1	1448
2002	800	96	10	15.1	2584
2003	850	64	10	15.1	2738
2004	550	32	5	12.1	999
2005	600	64	6	12.1	2399
2006	800	96	20	15.7	2999
2007	850	128	20	15.0	3099
2008	650	64	10	12.1	1249
2009	750	256	20	15.1	2599
2010	366	64	10	12.1	1499

(b) Экземпляр отношения Laptop

Printer

model	color	type	price
3001	true	ink-jet	231
3002	true	ink-jet	267
3003	false	laser	390
3004	true	ink- jet	439
3005	true	bubble	200
3006	true	laser	1999
3007	false	laser	350

(c) Экземпляр отношения Printer

Рис. 6.8. Отношения PC, Laptop и Printer

7. Реляционные операции на мультимножествах

Несмотря на то, что множество кортежей (т.е. отношение) является простой моделью данных в том их виде, в каком они могут входить в БД, коммерческие системы БД редко (и вряд ли когда-либо вообще) строятся исключительно на основе множеств. В некоторых ситуациях допускается, что отношения, входящие в систему БД, содержат дублирующиеся кортежи.

Напоминаем, что "множество", содержащее несколько вхождений какого-либо своего элемента, называется *мультимножеством.* В этом разделе рассматриваются отношения, являющиеся мультимножествами, а не множествами, т.е. один и тот же кортеж может входить в одно и то же отношение несколько раз. Термином "множество" обозначается отношение без повторяющихся кортежей, а термином "мультимножество" — отношение, содержащее (или не содержащее) повторяющиеся кортежи.

Пример. Отношение на рис. 7.1. — это мультимножество кортежей, в которое кортеж (1,2) входит трижды, а кортеж (3, 4) один раз. Если бы оно было основано на множестве, нужно было бы удалить из него два вхождения кортежа (1,2). В отношении, основанном на мультимножестве, действительно *допустимо* повторение вхождений некоторого кортежа, но, как и во множествах, порядок кортежей не имеет значения.

A	B
1	2
3	4
1	2
1	2

Рис. 7.1. Мультимножество

Причины использования мультимножеств

Анализ вопросов эффективной реализации отношений показывает, что использование мультимножеств может увеличить скорость операций на отношениях. Например, если мультимножество допускается в качестве результата проекции, с каждым кортежем можно работать независимо. Если же результатом является множество, необходимо сравнивать результат проекции из нежелательных компонентов каждого кортежа с результатом проецирования всех остальных кортежей, чтобы убедиться в том, что такая проекция ранее не встречалась. Если же результатом является мультимножество, мы просто проецируем каждый кортеж и добавляем его к результату; при этом не требуется никакого сравнения проекций.

A	*B*	C
1	2	5
3	4	6
1	2	7
1	2	8

Рис. 7.2. Мультимножество дли примере 7.1.

Пример 7.2. Мультимножество, изображенное на рис. 7.1, может быть результатом проекции отношения из рис. 7.2 на атрибуты *A и B* при условии, что результат — это мультимножество. Если же применить обычный оператор

проекции реляционной алгебры, а значит, удалить дубликаты кортежей, получается только отношение

Результат в виде мультимножества по размеру больше этого отношения, хотя вычисляется быстрее, так как необязательно сравнивать каждый кортеж (1,2) или (1,3) с построенными ранее кортежами.

Более того, если проекция отношения применяется для получения статистического значения типа среднего значения *A* в отношении, показанном на рис. 7.2, для анализа проецированного отношения *нельзя* применять модель множества. Для множества среднее значение *A* равно 2, так как .1 имеет только два различных значения: 1 и 3, средним значением которых является 2. Если же этот столбец на рис. 7.2. считается мультимножеством {1, 3, 1, 1}, получается правильное среднее значение *A* для четырех кортежей — 1,5.

Объединение, пересечение и разность мультимножеств

При объединении двух мультимножеств берется сумма вхождений каждого из кортежей. То есть, если *R*— мультимножество, в которое кортеж *t* входит *n* раз, а *S* — мультимножество, в которое кортеж *t* входит *m* раз, в мультимножество R U S кортеж t входит *n* + *m* раз. При этом *n и m* порознь или одновременно могут равняться 0.

При пересечении мультимножеств R и *S,* в которые кортеж *t* входит *n* и *m* раз, соответственно, в *R* ∩ *S* кортеж *t* входит min(n, *m)* раз. При вычислении разности *R* - *S* мультимножеств *R* и S кортеж *t* входит в *R* - *S* max(0, *n* - *m)* раз. То есть, если в *R* вхождений t больше, чем в S, то число вхождений t в *R*- *S* равно числу вхождений t в *R* минус число вхождений t в *S.* Если же число вхождений *t* в *S* не меньше числа вхождений t в *R,* то t вообще не входит в *R*- *S.* *C* интуитивной точки зрения каждое вхождение t в *S* "аннулирует" одно вхождение в R.

Мультимножественные операции на множествах

Представьте себе два множества— *R* и *S.* Любое множество может мыслиться как мультимножество, в котором случайно оказалось только по одному вхождению каждого элемента. Пусть выполняется пересечение *R* ∩ *S,* но *R* и *S* считаются мультимножествами и применяется правило пересечения мультимножеств- Результат получается такой же, как и тогда, когда *R* и *S* являются множествами. То есть, если *R* и *S* мыслятся как мультимножества, число «хождений кортежа t в *R* ∩ *S* равно минимальному из чисел его вхождений в *R* и *S.* Поскольку *R* и *S* на самом деле множества, t может входить в каждое из них только 0 или 1 раз. Независимо от того, применяются ли правила пересечения множеств или мультимножеств, оказывается, что t может входить в *R* ∩ *S* не более одного раза и входит в пересечение именно один раз, когда он входит и в R, и в *S.* Применение правила разности мультимножеств *R*- *S* или *S* - *R* тоже дает точно такой же результат, как и применение правила разности множеств.

Однако объединение ведет себя по-разному в зависимости от того, считаются ли *R* и S множествами или мультимножествами. Если для вычисления объединения *R U S* применяется правило для мультимножеств, результат может не быть множеством, даже если *R* и *S*— множества. В частности, если кортеж t входит в *R* и S, то он входит в *R U S* дважды, когда *R* и *S*— мультимножества. Если же применяется операция над множествами, *t* входит в *R U S* только один раз. Отсюда следует вывод: производя объединение, внимательно следите за тем, что именно объединяется - множества или мультимножества.

Проекция мультимножеств

Проекция мультимножеств уже рассматривалась в примере 7.1, где каждый кортеж обрабатывался независимо от других. Если R-мультимножество, изображенное на рис. 7.2, и вычисляется мультимножественная проекция $\pi_{A\ B}(R)$, получается мультимножество, показанное на рис. 7.1.

Если устранение одного или более атрибутов в процессе проекции вынуждает строить один и тот же кортеж из нескольких кортежей, повторяющиеся кортежи не удаляются из результата мультимножественной проекции. Значит, проекция каждою из кортежей (1, 2, 5), (1, 2, 7) и (1, 2, 8) отношения *R* из рис. 4.29 на атрибуты *A* и *B* порождает один и тот же кортеж (1,2). В результате-мультимножестве есть три вхождения этого кортежа, в то время как в результате-множестве только одно.

Алгебраические законы мультимножеств

Алгебраический закон - это эквивалентность двух выражений реляционной алгебры, аргументами которых являются переменные, обозначающие отношения. Суть эквивалентности в том, что два выражения определяют одно и то же отношение независимо от того, какие отношения подставлены вместо переменных. Пример широкоизвестного закона - закон коммутативности объединения: R U S = S U *R*. Он верен независимо от того, считаются ли реляционные переменные *R* и *S* множествами или мультимножествами. Однако есть ряд других законов, которые действуют, когда реляционная алгебра интерпретируется в обычном стиле - с отношениями в виде множеств, но не действуют, когда отношения интерпретируются как мультимножества. Простой пример такого закона — закон дистрибутивности теоретико-множественной разности относительно объединения: *(R U* S)- T=(R - T) U *(S - T)*. Он верен для множеств, но не для мультимножеств. Действительно, пусть *R, S* и *T*- мультимножества, содержащие по одной копии. Тогда в левой части эквивалентности есть одно вхождение t, а в правой нет ни одного. В случае с множествами ни в одной части эквивалентности нет вхождений *t.*

Операция выбора на мультимножествах

Для выполнения выбора на мультимножестве условие выбора применяется независимо к каждому кортежу и, как всегда с мультимножествами, повторяющиеся кортежи не удаляются из результата.

Произведение мультимножеств

Согласно правилу для декартова произведения мультимножеств, каждый кортеж одного отношения спаривается с каждым кортежем другого, независимо or того дублируется он или нет. В результате, если кортеж r входит в *R m* раз, а кортеж s входит в *S n* раз, кортеж rs войдет в *R* x *S mn* раз.

Соединения мультимножеств

Процедура соединения мультимножеств тоже не содержит сюрпризов. Каждый кортеж одного отношения сравниваем с каждым кортежем другого и проверяется, успешно ли они соединяются. Если да, полученный при их объединении кортеж вносится в результат, причем повторяющиеся кортежи из результата не устраняются.

8. Функциональные зависимости

Самый важный вид ограничений, применяемых в реляционной модели, ограничение по уникальности значений, называемое функциональной зависимостью. Знать такое ограничение просто необходимо для того, чтобы перерабатывать схемы БД с целью устранения избыточности. Существуют и другие ограничения, позволяющие создавать хорошие схемы БД: многозначные зависимости, ограничения существования и ограничения независимости.

Определение функциональной зависимости

Функциональная зависимость на отношении R —это утверждение вида:

Атрибут B функционально зависим от атрибута A реляционной таблицы, если значение A однозначно определяет значение B. Любые две строки, имеющие одинаковое значение атрибута A, будут также иметь одинаковые значения атрибута B. Также говорят, что **A *функционально определяет* B**. Эта зависимость выражается как

$$A \rightarrow B$$

Если B функционально зависит от A, то состояние таблицы определяет функцию B от A. Это не обязательно функция в том смысле, что существует некоторая формула или выражение, позволяющие вычислить значение B на основании значения A. Вместо этого, функциональная зависимость определяется строками таблицы. Чтобы найти значение атрибута B для некоторого значения a атрибута A, необходимо выполнить следующие действия.

1. Найти строку таблицы, в которой значение атрибута A равно ***a.***

2. Возвратить значение атрибута B этой строки.

Если состояние таблицы изменится, то функция, определяющая значение B на основании A, также может измениться.

Пример 8.1. Рассмотрим отношение Книги, экземпляр которого представлен на рис.8.1.

Название	Год	Объем	Тип	Автор	Издательство
Системы баз	2003	1080	Учебник	Ульман	Вильямс
Системы баз	2001	470	Учебник	Риккард	Вильямс
Системы баз	1988	400	Учебник	Ульман	Мир
Ассемблер	2004	600	Учебник	Юров	Питер
Незапертая дверь	2003	330	Детектив	Маринин	Эксмо
Visual Basic 6	2003	430	Справочн	Кузьменк	Бином

Рис. 8.1. Отношение *Книги*

В этом отношении можно установить множество функциональных зависимостей, например:

Название Год → Объем Тип

Название Год → Издательство

Поскольку левые части этих зависимостей совпадают, их можно записать одной строкой:

Название Год → Объем Тип Издательство

Неформально это множество зависимостей означает; если два кортежа имеют одно и то же значение в своих компонентах *Название* и одно и то же значение в компонентах *Год*, то они должны иметь одинаковые значения в компонентах Объем, одинаковые значения в компонентах *Тип* и одинаковые значения в компонентах *Издательство*. Такое утверждение имеет смысл, если вспомнить исходный проект, из которого была построена реляционная схема *Книги*. Атрибуты *Название* и *Год* образуют ключ для объектов из множества фильмов. Поэтому при наличии названия и года выпуска фильма, существуют также его уникальная длина, уникальный тип и единственная владеющая им студия.

В то же время утверждение

Название Год → Автор

ложно. Оно не является функциональной зависимостью, так как, согласно определению класса Книги, верно только то, что для каждой книги уникальным образом определено *множество* авторов..

Функциональные зависимости относятся к схемам

Помните, что функциональная зависимость, подобно любому ограничению, является утверждением о схеме отношения, а не об отдельном ее экземпляре. По отдельному экземпляру нельзя определить, верна ли для него функциональная зависимость. Например, глядя на рис. 3.27, можно предположить, что есть зависимость типа *Название* -» *Тип*, так как для

каждого кортежа в этом экземпляре отношения верно, что любые два кортежа, совпадающие по *Название*, совпадают и по *Тип*.

Ключи для отношений

Множество атрибутов *{A, B, C}* является *ключом* для отношения *R,* если:

1. Эти атрибуты функционально определяют все остальные атрибуты данного отношения, т.е. два различных кортежа *R* не могут совпадать по всем атрибутам A,B,C.

2. Ни одно собственное подмножество множества *{A,B,C}* функционально не определяет все остальные атрибуты *R,* т.е. ключ должен быть *минимальным.*

Если ключ состоит из единственного атрибута *A,* часто говорят, что *A* (а не *{A})* есть ключ.

Пример 8.1. Атрибуты {*Название, Год, Автор*} формируют ключ для отношения *Книги* на рис. 8.1. Во-первых, нужно показать, что они функционально определяют все остальные атрибуты. Допустим, два кортежа совпадают по этим трем атрибутам. Поскольку они совпадают по *Название* и *Год*, они должны совпадать и по остальным атрибутам: *Объем*, *Тип* и *Издательство*. Значит, два разных кортежа не могут совпадать по *Название*, *Год* и *Автор*, так как в случае такого совпадения они фактически были бы одним и тем же кортежем.

Теперь нужно доказать, что ни одно собственное подмножество множества { *Название*, *Год*, *Автор* } функционально не определяет все остальные атрибуты. Действительно, *Название* и *Год* не определяют *Автор*, поскольку многие книги могут писать несколько авторов. Значит, { *Название*, *Год* } не является ключом.

{*Год*, *Автор*} не является ключом потому, что один автор может написать несколько книг в одном и том же году. Следовательно,

Год Автор → *Название*

не является функциональной зависимостью. Можно также сказать, что { *Название*, *Автор* } не является ключом потому, что в разные годы могут быть написаны две книги с одним и тем же названием.

Иногда отношение имеет несколько ключей. В таком случае один из них считается *первичным ключом.* В коммерческих системах БД выбор первичного ключа может повлиять на вопросы реализации, например на способ хранения данного отношения на диске.

Суперключи

Множество атрибутов, содержащее ключ, называется супер*ключом* (сокращение выражения "надмножество ключа"). Значит, каждый ключ — это суперключ, а некоторые суперключи не являются (минимальными) ключами. Заметим, что любой суперключ удовлетворяет первому условию ключа: функционально определяет все остальные атрибуты отношения, но не должен удовлетворять второму условию минимальности.

Пример 8.2. В отношении из примера 8.1 много суперключей. Ими являются не только ключ {*Название*, *Год*, *Автор*}, но и любое надмножество этого множества атрибутов, например {*Название*, *Год*, *Автор*, *Объем*}.

Обнаружение ключей для отношений

Когда при конвертировании ODL или E/R-проекта в отношение строится ре ляционная схема, обычно можно заранее сказать, каким будет ключ этого ния. В данном разделе рассматриваются отношения, полученные из а в следующем — из ODL-проектов.

6 Помните, что функциональные зависимости — это допущения или утверждения, касающиеся данных. Не существует внешнего авторитета, способного выносить окончательное решение по поводу наличия или отсутствия функциональной зависимости. Поэтому в нашей ноле принимать любые разумные решения о том. какая функциональная зависимость Верн

Когда отношения строятся из E/R-проекта, в большинстве случаев (хотя и не всегда) для каждого отношения существует единственный ключ. Если отношение имеет единственный ключ, при составлении его схемы полезно подчеркивать атрибуты, которые этот ключ формируют.

Другая терминология, относящаяся к ключам

В некоторых книгах и статьях ключи описываются в другой терминологии. Иногда термином "ключ" обозначается то, что мы называем "суперключом", т.е. множество атрибутов, функционально определяющее все атрибуты без учета требования минимальности. В этих работах для минимального ключа (который мы обозначаем термином "ключ") обычно используется термин "потенциальный ключ" ("candidate key").

Первое правило введения ключей:

• Если отношение происходит из множества сущностей, ключом этого отношения являются ключевые атрибуты данного множества или класса.

Пример 8.3. В примерах 2.1 и 2.2 показано, как множества сущностей *Книги* и *Авторы* конвертируются в отношения. Ключами этих множеств являются *{Название, Год}* и *{Имя}* соответственно. Значит, они являются также ключами соответствующих отношений, имеющих следующие схемы:

Книги (Название, Год, Объем, Тип)

Авторы (Имя, Адрес)

Второе правило относится к бинарным связям. Если отношение *R* строится из связи, ее сложность влияет на ключ для *R*. Имеется три случая:

При связи типа "многие-ко-многим" ключи обоих связанных множеств являются ключевыми атрибутами для *R*.

Если есть связь типа "многие-к-одному" множества E_1 с множеством E_2, ключевыми атрибутами R являются ключевые атрибуты E_1 но не E_2-

При связи типа "один-к-одному" ключи любого, из связанных множеств являются ключевыми атрибутами для R. Поэтому для *R* не существует уникального ключа.

Другие понятия функциональной зависимости

Мы считаем, что в левой части функциональной зависимости может быть множество атрибутов, а в правой только один. Более того, атрибут, стоящий справа, не может появиться в левой части. Однако для краткости допускается комбинировать несколько зависимостей с одинаковыми левыми частями, чтобы получить множество атрибутов справа. Иногда полезна также "тривиальная" зависимость, в правой части которой стоит один из атрибутов, находящихся слева.

В других работах на эту тему часто принимается точка зрения, согласно которой левая и правая части зависимости являются произвольными множествами атрибутов и атрибуты могут находиться одновременно слева и справа. Между этими двумя подходами нет существенных различий, но мы будем исходить из того, что ни один атрибут не входит одновременно и правую и левую части функциональной зависимости, если явно не оговорено противоположное

Правила функциональной зависимости

В этом разделе показано, как *логически рассуждать* о функциональной зависимости. Допустим, нам предъявили множество зависимостей, которым удовлетворяет некоторое отношение. Часто из этого делается вывод о том, что данное отношение должно удовлетворять и другим конкретным зависимостям, при 'лом ; неизвестно даже, какие именно кортежи входят в это отношение. Такая способность открывать дополнительные зависимости играет существенную роль при рассмотрении проектов хороших схем в разделе 3.7.

Пример 8.4. Если отношение *R* с атрибутами *A, B* и С удовлетворяет функциональным зависимостям $A \rightarrow B$ и $B \rightarrow C$, можно сделать вывод, что *R* удовлетворяем также зависимости $A \rightarrow C$. Как же получается такой вывод? Чтобы доказать $A \rightarrow C$. рассмотрим два кортежа из *R*. совпадающих по *A,* и покажем, что они совпадают и по *C*.

Пусть кортежи *(a,* b_1, c_1,) и *(a,* b_2, c_2) совпадают по атрибуту *A,* а порядок атрибутов в кортежах — *A, B, C*. Поскольку *R* удовлетворяет $A \rightarrow B$ и кортежи совпадают по *A,* они совпадают и по *B*. Значит $b = b_2$ и кортежами действительно являются (a, b, c_1,) и (a, b, c_2), где *b* есть и b_1,, и b_2. Поскольку *R* удовлетворяет $B \rightarrow C$ и кортежи совпадают по *B,* они совпадают и по *C*. Значит, c_1, = c_2, т,е. кортежи действительно совпадают по С. Итак, мы доказали, что любые два кортежа R, совпадающие по А. совпадают и по С, а это и есть функциональная зависимость $A \rightarrow$ С.

Функциональную зависимость часто можно выразить различными способами, не изменяя множества допустимых экземпляров отношения; в таком случае говорят, что множества зависимостей *эквивалентны.* Считается,

что множество функциональных зависимостей *S следует* из множества функциональных зависимостей *T,* если каждый экземпляр отношения, удовлетворяющий всем зависимостям в T удовлетворяет и всем зависимостям в *S.* Заметим, что множества зависимостей *S* и *T* эквивалентны, если *S* следует из *T,* а *T*— из *S.*

В этом разделе рассмотрим множество полезных правил, касающихся функциональных зависимостей. Эти правила позволяют заменять множество зависимостей эквивалентным ему множеством или добавлять к множеству зависимостей другие множества, которые следуют из исходного множества. Примером таких правил является *правило транзитивности,* позволяющее следовать по цепи функциональных зависимостей, как в примере 3.26. Введем также алгоритм ответа на общий вопрос: следует ли функциональная зависимость из одной или нескольких других зависимостей.

Правило разделения/объединения

Мы определили функциональную зависимость

$A, A_2 ... A_n \rightarrow B_1 B_2 ... B_m$

как сокращение множества зависимостей.

Следовательно, можно расщепить атрибуты, стоящие в правой части, так, чтобы только один атрибут появлялся в правой части каждой функциональной зависимости. Можно также заменить множество зависимостей с общей левой частью единственной зависимостью с той же левой частью и всеми правыми частями, соединенными в одно множество атрибутов. В любом случае новое множество зависимостей эквивалентно исходному. Такую эквивалентность можно реализовать двумя способами.

Функциональную зависимость $A_1 A_2 \ldots A_n \rightarrow B_1 B_2 ... B_m$ можно заменить множеством функциональных зависимостей $A_1 A_2 \ldots A_n \rightarrow B_i$ для $i = 1, 2, ..., m$. Такое преобразование называется ***правилом разделения***.

Множество функциональных зависимостей $A_1 A_2 ... A_n \rightarrow B_i$ для $i = 1, 2, ... m$ можно заменить единственной функциональной зависимостью $A_1 A_2 \ldots A_n \rightarrow B_1 B_2 ... B_m$. Такое преобразование называется ***правилом соединения***.

Например, в примере 3.20 было показано, что множество зависимостей

Название Год → Объем
Название Год → Тип
Название Год → Издательство

эквивалентно единственной зависимости

Название Год → Объем Тип Издательство

Может показаться, что расщепление применимо и к левым частям функциональных зависимостей. Но следующий пример показывает, что правила расщепления левых частей не существует,

Пример 8.5. Рассмотрим зависимость

Название Год -> Объем

для отношения *Книги* из примера 3.20. Расщепив его левую часть, получим ложные зависимости

Название -> Объем

Год -> Объем

Другими словами, *Название* функционально не определяет *Объем*, так как могут существовать две книги различного объема, но с одним и тем же названием {например, *«Системы баз данных»).* Аналогично, *Год* функционально не определяет *Объем*. как и один год может быть выпущено множество книг различного объема.

Тривиальные зависимости

Функциональная зависимость $A_1A_2...A_n \rightarrow B$ называется *тривиальной,* если В совпадает с одним из *А.* Например, зависимость

Название Год -> Название тривиальна.

Любая тривиальная зависимость верна на любом отношении, так как это означает, что "два кортежа, совпадающие по всем атрибутам A_1, A_2, ..., A_n, совпадают по одному из них". Значит, можно допускать любую тривиальную зависимо не подтверждая ее знаниями о данных.

В исходном определении функциональных зависимостей их тривиальное п допускалась. Однако ничто не мешает использовать тривиальные зависимости, поскольку они всегда истинны и иногда упрощают формулировки правил,

Допуская тривиальные зависимости, мы допускаем и зависимости, в которых отдельные атрибуты находятся и справа, и слева. Зависимость $A_1 A_2... A_n \rightarrow B_1B_2... B_m$

- *тривиальна,* если $\{ B_1B_2... B_m \}$ является подмножеством множества $A_1 A_2... A_n$ '
- *нетривиальна,* если по крайней мере одно В не входит в множество $A_1 A_2... A_n$

полностью нетривиальна, если ни одно В, не совпадает ни с одним A_i

9. Вычисление замыкания атрибутов

Прежде чем перейти к другим правилам, рассмотрим общий принцип, из которого следуют все правила. Пусть $(A_1, A_2 ..., A_n\}$ есть множество атрибутов, а S — множество функциональных зависимостей. ***Замыкание (closure)*** $\{A_1, A_2, ..., A_n\}$по зависимостям в *S* есть такое множество атрибутов В, что любое отношение, удовлетворяющее всем зависимостям в *S,* удовлетворяет также зависимости $A_1 A_2... A_n \rightarrow B$. Значит, $A_1 A_2... A_n \rightarrow B$ следует из зависимостей *S.* Замыкание множества атрибутов $A_1, A_2, ..., A_n$ обозначается $\{A_1, A_2, ..., A_n\}^+$. Для того чтобы упростить ход рассуждений о вычислении замыканий, допускаются тривиальные зависимости, поэтому $A_1, A_2, ..., A_n$ всегда входят в $\{A_1, A_2, ..., A_n\}^+$.

Начиная с заданного множества атрибутов, будем последовательно расширять его, добавляя правые части функциональных зависимостей тогда, когда вводятся их левые части. Множество, которое больше

расширить уже нельзя, является ***замыканием***. Следующие шаги описывают алгоритм вычисления замыкания множества атрибутов *{*A*1, A2, ..., An,}* по отношению к множеству функциональных зависимостей.

1. Пусть *X*— множество атрибутов, которое в конечном счете станет замыканием. Начнем с того, что *X* есть *{*A*1, A2, ..., An}*.

2. Осуществляем поиск такой функциональной зависимости $B_1B_2... B_m \rightarrow C$, в которой все $B_1B_2... B_m$ входят в множество атрибутов *X,* а C нет. Затем добавляем *C* к множеству *X.*

3. Повторяем шаг 2 необходимое число раз, пока к множеству X уже невозможно будет добавить атрибуты. Поскольку X может только расти, а множество атрибутов любого отношения должно быть конечным, процесс завершается.

4. Множество *X,* к которому невозможно добавить атрибуты, и будет корректным значением замыкания $\{A_1, A_2, ..., A_n\}^+$.

Пример 9.1 Рассмотрим отношение с атрибутами *A, B, C, D, E* и *F*. Пусть оно имеет функциональные зависимости *AB*→ C, *BC*→ *AD, D* → *E* и *CF* → B. Вычислим замыкание **{A, B}**, т.е. $\{A, B\}^+$.

Начнем с *X = {A, B}*. Заметим сначала, что все атрибуты левой части функциональной зависимости *AB*→ *C* находятся в *X,* значит, можно добавить атрибут C расположенный в правой части этой зависимости. Таким образом, после повторения шага 2 *X* становится множеством *{A, B, C}*.

Теперь ясно, что левая часть зависимости *BC*→*AD* входит в *X,* поэтому к X можно добавить атрибуты A и D. A уже входит в *X,* но *D* не входит, значит, после добавления *X* становится множеством *{A, B, C, D}*. Теперь можно использовать зависимость *D*→ E, чтобы добавить *E* к X, после чего *X* становится множеств *{A, B, C, D, E}*. Дальнейшие изменения *X* невозможны. В частности, нельзя использовать функциональную зависимость *CF*→ *B,* так как ее левая часть никогда не войдет в *X.* Следовательно, $\{A, B\}^+ = \{A\ B, C, D\ E\}$.

Зная, как вычислять замыкание любого множества атрибутов, можем проверить, следует ли функциональная зависимость $A, A_2... A_n \rightarrow B$ из множества зависимостей *S.* Сначала вычисляем $\{A_1, A_2, ..., A_n\}^+$.с помощью множества зависимостей *S.* Если *B* входит в $\{A_1, A_2, ..., A_n\}^+$, тогда эта зависимость действительно следует из *S* . Если же *B* не входит в $\{A_1, A_2, ..., A_n\}^+$, данная зависимость не следует из *S.*

Правило транзитивности

Правило транзитивности позволяет строить новую функциональную зависимость из двух заданных.

Если A → B и B → C в отношении R, то верно и A → C в R

Замыкания и ключи

Заметим, что $\{A_1, A_2, ..., A_n\}^+$.является множеством всех атрибутов тогда и только тогда, когда $A_1, A_2, ..., A_n$— суперключ рассматриваемого отношения. Только в этом случае $A_1, A_2, ..., A_n$ функционально определяет все остальные атрибуты. Чтобы проверить, является ли $A_1, A_2, ..., A_n$ ключом отношения, достаточно сначала убедиться, что $\{A_1, A_2, ..., A_n\}^+$.— это все атрибуты, а затем установить, что ни для какого множества *S,* полученного удалением одного из атрибутов из $\{A_1, A_2, ..., A_n\}$ замыкание S^+ не является множеством всех атрибутов.

Замыкающие множества функциональных зависимостей

При наличии множества зависимостей часто можно вывести другие зависимости, в том числе тривиальные и нетривиальные. Необходимо различать ***заданные*** зависимости, которые первоначально устанавливаются для отношения, и ***производные*** зависимости, выведенные по правилам, описанным в этом разделе, или по алгоритму замыкания множества атрибутов.

Более того, иногда можно выбирать, какие зависимости использовать для представления полного множества зависимостей для отношения. Любое множество заданных зависимостей, из которого можно вывести все зависимости для отношения, называется ***базисом*** для этого отношения. Если полное множество зависимостей нельзя вывести ни из одного собственного подмножества зависимостей, входящих в базис, базис считается ***минимальным***.

Пример 9.1. Рассмотрим отношение *R (A, B,* C), в котором каждый атрибут функционально определяет все остальные атрибуты. Здесь полное множество выводимых зависимостей состоит из шести зависимостей, содержащих по одному атрибуту справа и слева: $A \rightarrow B$, $A \rightarrow$ C, $B \rightarrow A$, B--→ C, C→A и C → *B*. Оно содержит также три нетривиальные зависимости с двумя атрибутами слева каждая: $AB \rightarrow$ C, $AC \rightarrow B$ и BC→A B нем есть сокращения для пар зависимостей типа $A \rightarrow BC$, к тому же можно еще добавить тривиальные зависимости типа $A \rightarrow A$ или зависимости типа $AB \rightarrow$ BC, не являющиеся полностью тривиальными (хотя строгое определение функциональной зависимости и не требует перечисления тривиальных зависимостей, частично тривиальных зависимостей или зависимостей с множеством атрибутов в правых частях).

Такое отношение и его зависимости имеют множество минимальных базисов например:

{A → *B*, *B* → *A*, B → C, C → *B*} или {*A* → B, B → C, *C* → *A*}.

10. Разработка схем БД и декомпозиция

Как уже неоднократно отмечалось, при прямом преобразовании E/R-проектов в схемы реляционной БД возникают некоторые проблемы. Главная из них избыточность — повторение одного факта в нескольких кортежах.

Наиболее распространенная причина избыточности - соединение однозначных и многозначных свойств объекта в одном отношении. Например, избыточность возникает при попытке сохранить однозначную информацию о книге, например о ее объеме, вместе с многозначными свойствами типа множества авторов, пишущих книги. Такие проблемы просматриваются на рис. 10.1. Аналогичная избыточность возникает и тогда, когда мы пытаемся сохранить однозначную информацию о дате рождения автора вместе с его адресами.

Название	Год	Объем	Тип	Издательство	Автор
Системы баз данных	2003	1071	Учебник	Вильямс	Дж. Ульман
Системы баз данных	2003	1071	Учебник	Вильямс	Дж. Уидом
Системы баз данных	2003	1071	Учебник	Вильямс	Г. Молина
Visual Basic.NET	2003	450	Учебник	Питер	В. Долженков
Ассемблер	2004	635	Учебник	Питер	В. Юров
Незапертая дверь	2003	332	Детектив	Эксмо	А. Малинина

Рис. 10.1. Отношение *Книги* с аномалиями

Более подробно и последовательно рассмотрим задачи разработки хороших реляционных схем.

1 Сначала детально анализируются проблемы, возникающие при переполнении схемы.
2. Затем вводится понятие декомпозиции — разбиения схемы отношения (множества атрибутов) на более мелкие схемы.
3. Далее вводится нормальная форма Бойса-Кодда (BCNF) — условие для реляционной схемы, позволяющее решить упомянутые проблемы.
4. Два предыдущих пункта объединяются при объяснении того, как обеспечивается BCNF с помошью декомпозиции реляционных схем.

Аномалии

Проблемы типа избыточности, возникающей при попытке внести слишком большой объем информации в единственное отношение, называются *аномалиями.* основные виды аномалий:

1. *Избыточность.* Информация может без всякой необходимости повторяться в множестве кортежей. Примеры — *издательство* и *тип книги* на рис. 3.
2. *Аномалии обновления.* Информацию можно изменить в одном кортеже и оставить ее же без изменений в другом. Например, обнаружив, что книга *Системы баз данных* на самом деле имеет объем 1071 страниц можно по рассеянности изменить цифру в первом кортеже,. а во втором и третьем — нет. Конечно, можно заявить, что необходимо всегда быть

внимательным. Однако далее мы увидим, что невозможно перестроить отношение *Книги* так чтобы исключить риск подобной ошибки.

3. *Аномалии удаления.* Если множество значений становится пустым, побочным эффектом может быть утрата и другой информации. Например, при удалении В. Юрова из числа авторов книги *Ассемблер* в базе данных не останется больше авторов для этой книги. Последний кортеж для *Ассемблера* в отношении Книги исчезнет вместе с информацией о том, что данная книга учебник и имеет объем 635 страниц.

Декомпозиция отношений

Для устранения аномалий применяется метод *декомпозиции* отношений. Декомпозиция R — это расщепление атрибутов R для построения схем двух новых отношений. Правило декомпозиции включает в себя и способ заполнения новых отношении путем "проекции" кортежей отношения R. После описания процесса декомпозиции мы покажем, как выбирать декомпозицию, устраняющую аномалии.

Отношение R со схемой $\{A_1, A_2, ..., A_n\}$ можно *расчленить* на два таких отношения S и T *со* схемами $\{B_1, B_2, ... B_m\}$ и $\{C_1, C_2, ... C_k\}$ соответственно, что

1. $\{A_1, A_2, ..., A_n\}=\{Bi,B_2, ..-, B_m\}U\{C_1, C_2, ... C_k\}$
2. Кортежи в отношении S являются *проекциями* на $\{B_1, B_2, ... B_m\}$ всех кортежей в R. Другими словами, для каждого кортежа t в текущем экземпляре R берутся компоненты, соответствующие атрибутам $B_1, B_2, ... B_m$. Эти компоненты формируют кортеж, который и принадлежит текущему экземпляру отношения 5. Однако отношения являются множествами и один и тот же кортеж S может получиться из проекции двух различных кортежей R. В таком случае в текущий экземпляр S помещается только один экземпляр кортежа.
3. Кортежи в отношении Т--тоже проекции кортежей в текущем экземпляре R на множество атрибутов $\{C_1, C_2, ... C_k\}$

Пример. Выполним декомпозицию отношения *Книги*, показанного на рис. 9.1. Сначала расчленим схему. Наш выбор состоит в том, чтобы использовать:

1. Отношение *Книги1*. в схему которого входят все атрибуты, кроме Автор
2. Отношение Книги2, схема которого состоит из атрибутов Название, Год и Автор.

Затем рассмотрим процесс декомпозиции экземпляров отношений путем декомпозиции данных, приведенных на рис. 9.1. Сначала построим проекцию на схему Книги1:

(Название, Год, Объем, Тип, Издательство)

Первые три кортежи на рис. 10.1 имеют в этих пяти атрибутах одни и те же компоненты:

(Системы баз данных, 2003, 1071, Учебник, Вильямс)

Четвертый кортеж порождает для первых пяти компонентов совсем другой кортеж, а пятый и шестой кортежи порождают один и тот же пятикомпонентный кортеж. В результате получается отношение для Книги1, изображенное на рис. 10.2.

Название	Год	Объем	Тип	Издательство
Системы баз данных	2003	1071	Учебник	Вильямс
Visual Basic.NET	2003	450	Учебник	Питер
Ассемблер	2004	635	Учебник	Питер
Незапертая дверь	2003	332	Детектив	Эксмо

Рис. 10.2. Отношение *Книги1*

Теперь рассмотрим проекцию с рис. 9.1 на схему *Книги2*. Все шесть кортежей этой схемы отличаются друг от друга по крайней мере в одном из атрибутов Название, Имя или Автор, поэтому в результате получается отношение, показанное на рис. 10.3.

Название	Издательство	Автор
Системы баз данных	Вильямс	Дж. Ульман
Системы баз данных	Вильямс	Дж. Уидом
Системы баз данных	Вильямс	Г. Молина
Visual Basic.NET	Питер	В. Долженков
Ассемблер	Питер	В. Юров
Незапертая дверь	Эксмо	А. Малинина

Рис. 10.3. . Отношение *Книги2*

Заметим, что такая декомпозиция устраняет аномалии, упомянутые ранее. Исчезает избыточность. В частности, объем книги входит в отношение *Книги1* только один раз. Риск аномалии обновления тоже исчезает. Например поскольку нужно изменить объем книги *Системы баз данных* только в одном кортеже отношения *Книги1*, можно не опасаться, что появятся два различных объема этой книги.

И наконец, мы избавляемся от риска аномалии удаления. Так, при удалении всех авторов для книги *Ассемблер* эта книга исчезает из отношения *Книги2* но информацию о ней можно найти в отношении *Книги1*.

Может создаться впечатление, что отношению Книги2 все еще содержит избыточность, поскольку название книги и год выпуска появляются в нем многократно. Но эти два атрибута составляют ключ для множества книг, и более короткого способа представления книги не существует. Более того, Книги2 исключает возможность аномалии обновления.

Декомпозиция приводит к нормализации отношений. Рассмотрим нормальные формы отношений.

11. Нормальные формы

Нормализация — это процесс преобразования объектов к структурному виду, удовлетворяющему некоторому набору требований. Виды ***нормальных форм*** для реляционной схемы разработаны таким образом, что любая схема в нормальной форме гарантировано обладает определенными качественными характеристиками. Для определения разновидностей нормальной формы было проведено огромное количество исследований и экспериментов. В этом разделе рассматриваются те виды, которые относятся к преобразованию функциональных зависимостей в зависимости ключа. Они исключительно полезны для начинающих разработчиков баз данных. По мере усложнения схем баз данных возрастает роль других видов. Подробный список литературы по вопросам, касающимся нормальных форм, приводится в разделе "Дополнительная литература" в конце главы.

В данном разделе рассмотрены четыре вида нормальных форм: первая нормальная форма (1NF), вторая нормальная форма (2NF), третья нормальная форма (3NF) и нормальная форма Бойса—Кодда (BCNF). Каждое из последующих правил дополняет предыдущее. В результате приведенная к BCNF схема базы данных обычно находится в виде, подходящем для создания системы с реляционной базой данных.

Для каждой нормальной формы существует правило, описывающее, какие виды функциональных зависимостей в ней допускаются. Нормализация — это процесс преобразования схем с целью устранения нарушений правил нормальной формы. Нормализация применяется независимо к каждой реляционной схеме из схемы базы данных. Говорят, что схема базы данных приведена к *нормальной форме,* если каждая из ее реляционных схем приведена к нормальной форме.

Не каждую функциональную зависимость следует рассматривать как потенциальное нарушение нормальной формы. Важнейшим общим правилом является рассматривать только зависимости, имеющие минимальное множество атрибутов в левой части, максимальное множество атрибутов в правой части и не имеющие атрибутов, входящих одновременно в обе части. Более точно определить смысл вышесказанного можно будет после введения некоторых обозначений.

Замечание о декомпозиции схемы

Нарушения требований нормальной формы можно устранить из схемы посредством ее декомпозиции на две схемы. Предположим, что схема S: (A, B, C, D, E, F) с ключом {A, B, C} содержит функциональную зависимость {A, B} →• {E, F}.

В процессе декомпозиции правые атрибуты зависимости {E, F} удаляются из исходной схемы и создается новая основная схема S: (A, B, C, D). Все атрибуты зависимости объединяются в новой зависимой схеме R: (A, B, E, F). Левые атрибуты зависимости формируют ключ новой схемы. Эти атрибуты остаются также в основной схеме в качестве внешнего ключа к новой схеме. Примеры приводятся в следующих разделах.

Зависимость X → Y называется полной зависимостью, если Y не является зависимым ни от какого подмножества X, т.е. ни для какого подмножества A из X не может существовать функциональной зависимости X - {A} → Y. Естественно, что если X состоит из одного-единственного атрибута, то X → Y является полной зависимостью. Полная функциональная зависимость имеет минимальную левую часть.

Зависимость X → Y называется *максимальной зависимостью,* если ни в X, ни в Y не существует атрибута A такого, что X → Y U {A}, т.е. ни один дополнительный атрибут не может быть добавлен к правой части зависимости.

Зависимость X → Y называется *тривиальной зависимостью,* если по меньшей мере один атрибут входит в обе части; т.е. пересечение X ∩ Y не пусто.

Без потери общности может быть сделано следующее ограничение. Только полные, максимальные и нетривиальные зависимости должны рассматриваться как нарушения нормальной формы.

Первая нормальная форма: атомарные атрибуты

В исторической ретроспективе сначала следует рассмотреть *первую нормальную форму* (1NF). В этой форме требуется, чтобы значение каждого атрибута схемы принадлежало атомарному домену. На некотором этапе развития реляционной модели форма 1NF была включена в основное определение.

Вторая нормальная форма: отсутствие зависимостей частичного ключа

Вторая нормальная форма разработана для того, чтобы исключить функциональные зависимости, содержащие в левой части какую-либо часть ключа.

Реляционная схема находится во *второй нормальной форме* (2NF), если отсутствуют непервичные атрибуты, частично зависящие от любого из ключей схемы. Другими словами, функциональная зависимость нарушает 2NF, если атрибуты ее левой части
образуют собственное подмножество некоторого ключа, а все атрибуты правой части являются непервичными.

Более формально, функциональная зависимость X → Y является *частичной зависимостью от ключа* схемы R, если существует набор атрибутов W, принадлежащий X такой, что W является ключом R, т.е. схема находится в 2NF, если в ней не существует частичных зависимостей от ключа.

Третья нормальная форма: отсутствие транзитивных зависимостей

Преобразование *Информация_ о_поставках* не устранило всех неключевых зависимостей. Схема S2 имеет следующие неключевые зависимости.

Код_поставщика → (Имя_поставщика, Улица,
Город, Штат, Почтовый_код}
Почтовый_код →• (Город, Штат }
{ Улица, Город, Штат } → Почтовый_код

Все эти зависимости являются нарушениями требований третьей нормальной формы. Каждая из них является неключевой зависимостью, т.е. зависимостью, в которой непервичные атрибуты зависят от атрибутов, не содержащих ключ. Каждая из этих зависимостей является транзитивной, поскольку ключ схемы определяет неключевое множество атрибутов, которое, в свою очередь, определяет набор непервичных атрибутов.

Функциональная зависимость X → Y является *транзитивной зависимостью,* если существует множество атрибутов Z, не являющееся подмножеством ни одного из ключей, для которого выполняется: X → Z и Z → Y.

Считается, что схема находится в *третьей нормальной форме (3NF),* если она находится в 2NF и не имеет транзитивных зависимостей. Нарушения 3NF устраняются с помощью того же правила декомпозиции, что и нарушения 2NF.

Нормальная форма Бойса-Кодда: отсутствие неключевых зависимостей

Нормальная форма Бойса-Кодда (BCNF - Boyce-Codd Normal Form). имеет наиболее простое определение среди нормальных форм. Схема находится в BCNF, если она не содержит неключевых зависимостей. BCNF усиливает 3NF требованием, что первичные атрибуты не могут зависеть от неключевых атрибутов. BCNF предназначена для обработки зависимостей, возникающих в схеме, имеющей два (или более) составных ключа, содержащих по меньшей мере один общий атрибут.

Рассмотрим более формально процесс нормализации в BCNF.

Декомпозиция предназначена для замены отношения несколькими новыми отношениями, свободными от аномалий. Оказывается, есть простое условие, при котором аномалии изначально не могут существовать.

- Отношение *R* есть BCNF. если и только если при наличии в R нетривиальной зависимости A_1, A_2, ..., A_n → *B* множество {A_1, A_2, ..., A_n} является суперключом для *R*.

Это значит, что левая часть любой нетривиальной зависимости должна быть ключом. Напоминаем, что суперключ не всегда минимален. Поэтому BCNF эквивалентна условию, согласно которому левая часть любой нетривиальной функциональной зависимости должна содержать ключ.

12. Работа с данными из базы

Мы уже можем выбирать информацию из базы данных с помощью операций реляционной алгебры. Оператор $\pi_L(\sigma_C(R))$ является основным в реляционной алгебре. Здесь **L** – список атрибутов, **C** – условие, **R** – имя отношения. На основе операций реляционной алгебры разработан язык **SQL** (***Structured Query Language***), который позволяет выполнять операции над данными в базе данных, которая находится на компьютере.

Оператор SELECT без преувеличения является главным в SQL. Он позволяет выполнять поиск, выбор и представлять данные самыми разными способами. С его помощью можно ответить на вопросы о типе, количестве и расположении данных.

Стандартный простейший набор операторов – это предложения **SELECT, FROM и WHERE**, условия и выражения.

Все дальнейшие усложнения начинаются со следующей конструкции оператора SELECT:

SELECT список атрибутов ***FROM*** таблица(ы) [***WHERE*** условия]

В списке атрибутов указывается, из каких столбцов нужно выбрать данные. В списке таблиц определяется, в каких таблицах содержатся эти столбцы. В предложении WHERE указывается условие выбора строк таблицы. Предложения SELECT и WHERE могут содержать в себе как константы, так и выражения. С помощью комбинаций предложений SELECT, FROM и WHERE вы сможете получить ответы на интересующие вас вопросы.

Важно понять, что результатом выполнения запроса является таблица, в которой названия столбцов соответствуют списку атрибутов, а строки отобраны в соответствии с условием.

Кроме этого к оператору можно добавить ключевые слова **ORDER BY** и **DISTINCT**, а также агрегирующие функции. Рассмотрим также предложения **GROUP BY** и **HAVING**. Коснемся также многотабличных запросов и вопросов объединения таблиц. Будут рассмотрены **вложенные запросы,** также известные как подзапросы.

На рисунке 10.1 представлена база данных, состоящая из двух таблиц: ***Сотрудники (Employees)*** и ***Отделы (Department)***. Данные в таблицах имеют разный тип: символьный (*Character*), числовой (*Numeric*), дата (*Date*), денежный (*Currency*), логический (*Logical*) и некоторые другие. В разных системах этот набор может несколько меняться, но основные типы сохраняются.

Данные, выбираемые с помощью оператора SELECT, находятся на пересечении предложений SELECT (столбец) и WHERE (строка). Давайте рассмотрим использование оператора SELECT на примере таблицы *Employees (Сотрудники)*.

В таблице Сотрудники (***Employees)*** хранится информация о сотрудниках: фамилии (*Lastname*), зарплата (*Salary*), адреса (*Address*), даты рождения (*Birthdate*), даты приема на работу (*Hiredate*), номер отдела (*Deptid*), и телефон (*Telephone*).

В таблице Отделы ***(Department***) хранится информация об отделах, в которых работают сотрудники: Номер отдела (*Deptid*), Название отдела (*Depname*), Этаж (*Depfloor*), на котором находится отдел и фамилия руководителя (*Manager*)

Самый простой вариант использования оператора SELECT – просмотр содержимого таблицы базы.

SELECT * FROM *Employees*

Звездочка означает, что из таблицы выбираются все столбцы

Результат выполнения оператора – это вывод всех данных из таблицы *Сотрудники*

Запрос 1. Чтобы узнать фамилии Сотрудников, живущих на Фестивальной улице (без адресов и дат), нужно указать список соответствующих столбцов и использовать предложение WHERE для указания условия отбора данных из таблицы, возвращаемых оператором SELECT.

Employees

Lastname	Salary	Birthdate	Hiredate	Deptid	Address	Telephone
Антонов	1120.0000	01/02/77	10/02/90	101	Festivalnaya	8-916-2345678
Петров	2320.0000	12/06/66	10/02/98	101	Festivalnaya	452-1895
Котов	2049.7400	06/16/60	11/12/90	102	Pulkovskaya	459-1234
Мотов	1430.0000	10/22/70	06/18/99	102	Avangardnaya	8-916-5-3767
Сыров	1155.0000	10/23/65	03/15/92	103	Narva	459-0989
Фоменко	1210.0000	08/18/82	01/01/04	103	Narva	459-1256
Кощкин	1155.0000	09/11/77	03/03/99	104	Flotskaya	454-6745
Пирогов	2650.0000	04/04/66	01/01/99	101	Flotskaya	455-6789
Kitov	1155.0000	10/25/66	03/12/99	104	Flotskaya	452- 3489
Круглов	2330.0000	09/10/77	01/15/95	102	Narva	459-9876
Дронов	12000.0000	11/10/77	07/22/99	104	Pulkovskaya	8-903-5550303

Department

Deptid	Depname	Depfloor	Manager
101	sales	1	Шифрин
102	marketing	1	Бубин
103	program	2	Иванов
104	store	2	Сидоров
105	Столовая	3	Фридман

Рис. 12.1. База данных Сотрудник и Отделы

Символьные данные всегда указываются в апострофах.

SELECT Lastname FROM Employees WHERE Address = ‘Festivalnaya’

Ответом будет таблица со списком сотрудников, живущих на Фестивальной улице:

Антонов

Петров

Запрос 2. Найдем сотрудников родившихся в 1977 году

Для этого используется встроенная функция YEAR( ). С ее помощью мы из поля даты выберем только год

SELECT Lastname FROM employees WHERE YEAR(Birthdate) = 1977

Ответ список сотрудников:

LastName

Антонов
Кошкин
Круглов
Дронов

Запрос 3. Найдем сотрудников, родившихся в Октябре:

Для этого используется встроенная функция MONTH ( ). С ее помощью мы из поля даты выберем только месяц.

SELECT Lastname FROM employees WHERE MONTH(Birthdate) = 10

Ответ список сотрудников:

LastName
Мотов
Сыров
Китов

Запрос 4. Найти сотрудников принятых на работу между 1990 и 1992 годами. Диапазон данных в SQL указывается с помощью слов BETWEEN и AND.

SELECT Lastname FROM employees WHERE Year(Hiredate) BETWEEN 1990 AND 1992

Ответ:

LastName
Антонов
Котов
Сыров

Запрос 5. При выводе данных названия полей можно переименовать. Тогда в выводимой таблице столбец будет называться так как он указан после ключевого слова AS:

SELECT Lastname **AS Фамилия**, Hiredate **AS Дата_приема** FROM employees WHERE YEAR(Hiredate) BETWEEN 1990 AND 1992

Ответ:

Фамилия ***Дата_Приема***
Антонов 10/02/90
Котов 11/02/90
Сыров 03/15/92

Запрос 6. Найти названия отделов, расположенных на втором этаже.

SELECT Depname, Depfloor FROM Department WHERE Depfloor = 2

Ответ:

Depname *Depfloor*

Program 2
Store 2

Запрос 7. Найти сотрудников с мобильными телефонами с кодом ‘8-916’. Для этого используется ключевое слово **LIKE,** которое позволяет найти все строки с одинаковой частью данных - подстрокой. Остальная часть строки заменяется знаком %.

SELECT Lastname FROM Employees WHERE telephone **LIKE** '8-916%'

Ответ:
Lastname
Антонов
Мотов

Запрос 8. Найти всех сотрудников, живущих на Нарвской или Флотской улицах. Если в запросе участвуют два условия, они могут быть соединены словами AND, OR или NOT. В нашем случае это **OR** (или).

SELECT Lastname FROM employees WHERE Address = 'Narva' **OR** Address = 'Flotskaya'

Ответ: фамилии, живущих на этих улицах людей (6 человек).

Работа с несколькими таблицами

Все рассмотренные выше запросы выполнялись над одной таблицей. Часто приходится выполнять запросы над двумя и больше таблиц. В этом случае их нужно соединить по общему полю.

Запрос 9. Найти сотрудников отдела, где начальник Шифрин. Так как фамилии сотрудников находятся в таблице *Сотрудники*, а фамилия начальника в таблице *Отдел,* мы должны соединить две таблицы. Соединение можно выполнить по общему полю *deptid.* Второе условие состоит в выборе только тех строк, в которых начальником (*Manager*), является Шифрин. Эти два условия соединяются словом **AND**.

SELECT Lastname, Manager FROM employees, Department WHERE employees.deptid = Department.deptid **AND** Manager = 'Шифрин'

Ответ:

Lastname	*Manager*
Антонов	Шифрин
Петров	Шифрин

Пирогов Шифрин

Запрос 10. Найти всех сотрудников, работающих на втором этаже.

SELECT Lastname FROM employees, Department WHERE employees.deptid = Department.deptid AND Depfloor =2

Ответ:
Lastname
Сыров
Фоменко
Кошкин
Китов
Дронов

Вложенные запросы

Мощным средством языка SQL являются вложенные запросы или подзапросы (***subquery***).

Подзапрос (subquery) — это дополнительный метод манипуляций с несколькими таблицами. Это — оператор SELECT, вложенный:

- в предложение WHERE, HAVING или SELECT другого оператора SELECT;
- в оператор INSERT, UPDATE или DELETE;
- в другой подзапрос.

Именно возможность вложения операторов SQL друг в друга является причиной, по которой SQL первоначально был назван Structured Query Language (язык структурированных запросов). Термин *подзапрос* часто используется для ссылки на всю совокупность операторов, которая включает один или несколько подзапросов, а также на отдельное вложение. Каждый включающий оператор — следующий по старшинству уровень в подзапросе — представляет собой внешний уровень для внутреннего подзапроса.

•Подзапросы — достаточно сложная тема, поскольку существует два типа их обработки — некоррелированная и коррелированная -- и три возможности соединения подзапроса с внешним предложением.

Упрощенная форма синтаксиса подзапроса иллюстрирует вложение подзапроса в операторе SELECT.

Подзапросы возвращают результаты внутреннего запроса во внешнее предложение. Подзапросы бывают трех типов, в зависимости от элементов в предложении WHERE внешнего запроса.

1. Подзапросы, которые не возвращают ни одного или возвращают несколько элементов (начинаются с IN или с оператора сравнения, содержат ключевые слова ANY или ALL).
2. Подзапросы, которые возвращают единственное значение (начинаются с

простого оператора сравнения).

3. Подзапросы, которые представляют собой тест на существование (начинаются с EXISTS).

Важно помнить, что в подзапросе можно задавать только одно имя столбца, то есть подзапрос возвращает в качестве результата всегда только один столбец (или как частный случай только одно значение).

Запрос 13. Найти сотрудников отдела, где начальник Шифрин. *Запрос 9* может быть выполнен другим способом – с помощью вложенного запроса. Внутренний запрос (подзапрос) находит номер отдела (Deptid) из таблицы Отдел, где начальник Шифрин и передает его во внешний (основной) запрос, который и находит всех сотрудников с этим номером отдела. Результат такой же, как и в запросе 9.

SELECT lastname AS family, e.deptid FROM employees e WHERE deptid **IN** (SELECT deptid FROM department WHERE manager = 'Шифрин')

Запрос 14. Найти сотрудников, живущих на той же улице, что и Китов. Так как подзапрос возвращает единственное значение, то в условии *where* основного запроса мы можем использовать знак равенства =.

SELECT lastname, deptid, address FROM employees WHERE address = (SELECT address FROM employees WHERE lastname = 'Kitov')

Ответ:

Lastname	*Deptid*	*Address*
Кошкин	104	Flotskaya
Пирогов	101	Flotskaya
Китов	104	Flotskaya

Агрегирующие функции

Агрегирующие функции используются для получения обобщающих значений. Их можно применять к **наборам (set)** строк: ко всем строкам таблицы, строкам, определенным в предложении WHERE, или к группам строк в предложении GROUP BY (это предложение описывается позже). В любом случае, независимо от структуры набора строк, *для каждого из них получается единственное значение.*

Обратите внимание на различие в результатах выполнения следующих запросов: первый находит для каждой строки таблицы *Сотрудники* соответствующее значение зарплаты, второй вычисляет общую сумму зарплаты (одно значение для набора, состоящего из всех строк таблицы).

SQL:

```
select sum(salary) from employees
31801.7140
```

```
select sum(distinct salary) as total from employees
29491.7140
select count(salary), count(distinct salary) from employees
```

Ответ:

count(salary)	count(distinct salary)
11	9

Агрегирующие функции всегда имеют аргументы.

Агрегирующая функция	*Результат*
SUM ([Distinct]выражение)	Сумма (различных) значений
AVG ([Distinct]выражение)	Средняя величина (различных) значений
COUNT([Distinct]выражение)	Количество (различных) ненулевых значений
COUNT (*)	Количество выбранных строк
MAX (выражение)	Максимальное значение
MIN (выражение)	Минимальное значение

В качестве аргументов обычно используются имена столбцов, но допускается использование констант, функций и их комбинации с арифметическими операторами.
Distinct означает, что в вычисление включаются только различные значения.

Запрос 15. Подсчитать количество строк в таблице Сотрудники
select count(*) from employees
Ответ: 11

Запрос 16. Найти всех сотрудников, получающих минимальную зарплату/
Используем подзапрос, который возвращает значение минимальной зарплаты в основной запрос, который, в свою очередь находит сотрудников с этим значением.

select lastname, deptid, salary from employees where salary = (select min(salary) from employees)

Ответ

Lastname	Deptid	Salary
Антонов	101	1120
Фоменко	103	1120

Группировка данных

На практике агрегирующие функции чаще используются в комбинации с предложением GROUP BY.

Рассмотрим предложение GROUP BY, которое возвращает в результате группы строк, и предложение HAVING, с помощью которого можно задать условия отбора в предложении GROUP BY.

Предложение GROUP BY неразрывно связано с агрегирующими функциями, без них оно практически не используется. Предложение GROUP BY разделяет таблицу на наборы, а агрегирующая функция вычисляет для каждого из них итоговое значение. Эти значения называются агрегирующим вектором. (В предыдущей главе рассматривались скалярные агрегирующие функции, которые в результате возвращали единственное значение.) На рис. 6.1 представлены примеры запросов, в которых использованы скалярные и векторные агрегирующие функции.

В контексте оператора SELECT предложение GROUP BY имеет следующий вид:

SELECT список_выбора FROM список_таблиц
[WHERE условия]
[GROUP BY список_группировки]
[ORDER by список_порядка]

В большинстве диалектов SQL каждый элемент из списка GROUP BY должен обязательно присутствовать в списке выбора — другими словами, группировать можно только выбираемые элементы. Разные системы, кроме имен столбцов, позволяют использовать в списке группировки выражения, заголовки столбцов и их порядковые номера в списке выбора.

Запрос 17. Найти средние зарплаты по отделам.

SELECT avg(salary), deptid FROM Employees GROUP BY deptid

Запрос 18. Найти сколько сотрудников работает на каждом этаже.

SELECT Count(lastname) AS Кол_во, Depfloor AS Этаж FROM employees, Department WHERE employees.deptid = Department.deptid GROUP BY Depfloor
Ответ:
Ответ:

Кол-во	Этаж
6	1
5	2

Кроме выборки мы можем добавить, изменить или удалить данные в таблицах базы данных с помощью операторов INSERT, UPDATE и DELETE.

Запрос 19. .Вставить новую запись о Кротове. Если мы не знаем на данный момент всех данных о Дронове, мы можем вставить только часть данных. В данном случае только фамилию (lastname) и зарплату (salary) и указываем их значения после ключевого слова VALUES.

INSERT INTO employees (lastname, salary) VALUES ('Кротов', 12000)

Запрос 20. Увеличить зарплату на 10% сотрудникам, принятым на работу до 1991 года. Новое значение указывается как выражение после ключевого слова SET, а год выбирается из даты приема на работу (Hiredate) с помощью функции YEAR(Hiredate).

UPDATE employees SET salary = salary + salary * 0.1 WHERE YEAR(hiredate) < 1991

Запрос 21 Шифрин повышает зарплату всем своим сотрудникам на 1000. Для этого используется подзапрос, в котором определяется номер отдела Шифрина, который возвращается в главный запрос.

UPDATE employees SET salary = salary + 1000 WHERE deptid = (SELECT deptid FROM department WHERE manager ='Шифрин')

Запрос 22. Удалить запись с данными о Дронове из таблицы Сотрудники.

DELETE FROM Employees WHERE Lastname = ‘Дронов’

Создание и использование виртуальных таблиц (курсоров)

Подобно операции объединения, курсоры являются неотъемлемой частью реляционной модели. Курсор создается с помощью оператора SELECT и обеспечивает гибкость при анализе и обработке данных. Курсор можно представлять себе как подвижный кадр или окно, через которое пользователь видит данные.

В предыдущих главах было продемонстрировано, как пользоваться оператором SELECT для выбора строк, комбинирования таблиц, переименования столбцов и выполнения вычислений, чтобы получить нужную вам информацию в конкретном виде, Создание курсора с помощью оператора SELECT позволяет вам без труда анализировать и обрабатывать именно те данные, которые вам (или кому-то другому) нужны-ни больше ни меньше.

Курсоры — это не какие-то отдельные копии данных из таблицы (таблиц) или другого курсора (курсоров), из которых они получены. Курсоры часто называют виртуальными таблицами, поскольку они не существуют как независимые объекты в базе данных (как, например, "настоящие" таблицы). (Термину "курсор" в ANSI соответствует термин **просматриваемая таблица (viewed table),** а "настоящая" таблица базы данных называется **базовой таблицей (base table).)** Запросы к курсорам в основном выполняются так же, как и к таблицам. Однако модификация данных посредством курсоров носит ограниченный характер.

Курсор определяется в операторе SELECT. Когда пользователь обращается к тому или иному курсору, система баз данных ассоциирует с ним соответствующие данные. Курсор находится на вершине этого процесса, скрывая от пользователя все его технические подробности. Прелесть курсора заключается в его простоте и универсальности: начинающих пользователей не пугают всевозможные объединения, опытные же пользователи не испытывают искушения манипулировать с данными, до которых им, вообще говоря, нет никакого дела, а нетерпеливым пользователям не приходится замедлять свою работу из-за необходимости ввода длинных операторов SQL.
Вот упрощенный синтаксис оператора, определяющего курсор:
CREATE VIEW имя_курсора [(имя_столбца [, имя_столбца]. . .)] AS SELECT_оператор

Запрос 22. Создать курсор, содержащий сотрудников отдела 101.

CREATE VIEW Primer AS SELECT lastname FROM employees WHERE deptid = 101
SELECT * FROM Primer

Упражнения.

Поставьте на компьютер Систему управления базами данных Visual FoxPro 6.0 или выше. Создайте базу данных (смотри пособие по выполнению лабораторных работ). При запуске Visual FoxPro 6.0 на экране открывается командное окно (Command Window), в котором вы можете набирать команды SQL.

13. Системы управления реляционными базами данных

Для создания базы данных, изменения ее структуры, редактирования и выборки данных используются специальные программные продукты - системы управления базами данных (СУБД). Современные реляционные системы управления базами данных содержат:

- набор инструментов для создания таблиц и отношений между связанными таблицами;
- средства администрирования базы данных;
- развитый пользовательский интерфейс, который позволяет получить доступ к информации, хранящейся в базе данных;
- средства разработки приложений, использующих базы данных.

С помощью средств СУБД вы можете:

- выбрать информацию, представляющую для вас интерес. Например, вы можете получить сведения обо всех междугородних разговорах определенного клиента за любой интервал времени;
- вывести на печать всю таблицу или только выбранные записи и поля в различных форматах.

- отображать информацию базы данных в графическом виде.
- осуществлять необходимые вычисления при формировании отчетов и выборке данных из таблиц.

В настоящее время имеется около десяти популярных СУБД для персональных компьютеров. Одними из самых популярных традиционно являются Visual FoxPro версии 6.0 и выше, а также СУБД, являющаяся частью Microsoft Office, – MS Access.

Visual FoxPro является удобной, развитой, постоянно обновляемой СУБД, позволяющей создавать как простые, так и большие базы данных, автономные и в архитектуре клиент – сервер. Лабораторные работы выполняются в среде Visual FoxPro.

Рассмотрим кратко возможности и особенности работы с СУБД Access.

Access — это гибкая и мощная система, позволяющая работать как с простыми, так и со сложными базами данных. Следует добавить, что это реляционная база данных, то есть база данных, которая позволяет определять отношения между различными категориями информации (как, например, между данными об отделах и данными об их сотрудниках). В результате вы имеете возможность пользоваться данными совместно.

Начало работы

Чтобы запустить Access, щелкните на кнопке Microsoft Access 2000 на панели инструментов Microsoft Office или на кнопке Пуск (*Start*) на панели задач и выберите команду Программы > Microsoft Access (*Programs > Microsoft Access*). Когда Access начнет работу, вы увидите окно диалога, показанное на рис. 1. Список имен, перечисленных в нижней части окна, будет меняться в зависимости от того, какие базы данных или проекты вы создали ранее. На данном этапе у вас есть выбор:

- создать новую базу данных, полагаясь на свои знания, или воспользоваться помощью Мастера баз данных (Database Wizard);
- создать проект Access;
- открыть базу данных или проект, которые были созданы ранее.

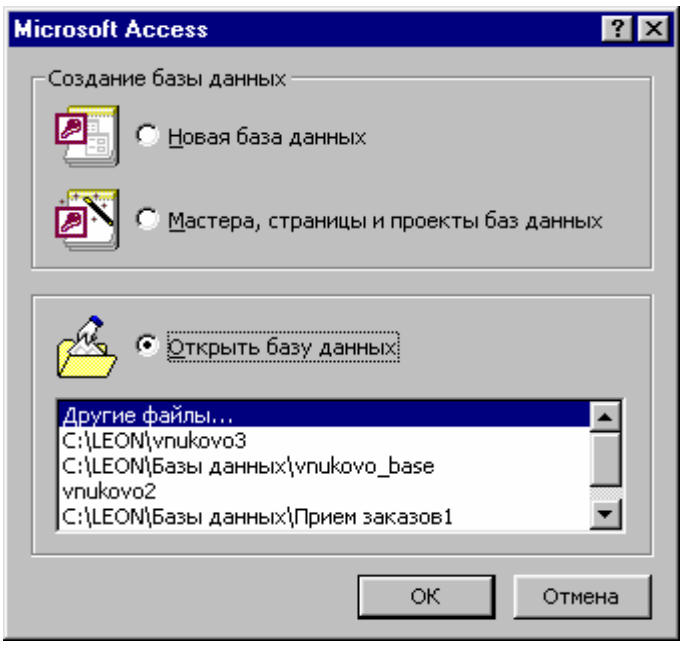


Рис. 13.1. Начальное окно диалога Access позволяет создать новый файл базы данных Access или открыть файл, с которым вы уже работали

Компоненты базы данных Access

Основу базы данных составляют хранящиеся в ней данные. Однако в базе данных Access есть и другие важные компоненты, которые принято называть объектами. Ниже приводится список типов объектов, с которыми вы будете работать.

1. **Таблицы** — содержат данные.
2. **Запросы** — позволяют задавать условия для отбора данных и вносить изменения в данные
3. **Формы** — позволяют просматривать и редактировать информацию.
4. **Отчеты** — позволяют обобщать и распечатывать информацию.

Таблицы

Любая информация, которую вы заносите в базу данных Access, сохраняется в ***таблице***. Таблица состоит из строк и столбцов. На языке базы данных строки называются **записями**, а столбцы — **полями**.

В Access существует множество различных способов создания таблицы. Все эти способы рассматриваются позже.

Запросы

Запросы предназначены, прежде всего, для отбора данных. Вы можете создать простой запрос для поиска записей в одной таблице, а можете сформулировать сложный запрос, включающий в себя данные из нескольких таблиц и учитывающий множество условий отбора. Ниже перечислены только некоторые вопросы, ответы на которые можно найти с помощью запросов.

- Какие сотрудники были приняты на работу за последние три месяца?

- Каково количество новых заказчиков, к которым я обращался на последней неделе?
- Каковы показатели по средне зарплате по отделам

Запрос на выборку

Продемонстрируем довольно простой запрос: с помощью него мы находим все записи в таблице Отделы для Ивана Петрова. Такой тип запроса называется *запросом па выборку (select query).* Он предназначен для поиска записей, удовлетворяющих сформулированными вами условиям отбора записей.

Позже мы более подробно остановимся на функциях запросов на выборку и рассмотрим несколько конкретных примеров работы в окне конструктора запросов (Query Design).

Другие типы запросов

Возможности конструктора запросов не ограничены созданием только одного типа запросов — запросов на выборку. Запросы можно использовать не только для поиска записей, но и для внесения в них изменений. С помощью запроса можно создать перекрестное представление данных (путем создания *перекрестного запроса),* создать таблицу, удалить из нее определенные записи или же добавить записи в одну таблицу из другой. Существуют следующие типы запросов.

Перекрестный запрос**.**

С помощью запросов этого типа вы можете сделать обзор по категориям данных, то есть обобщить информацию. Например, можно выяснить, каков был объем продаж по каждому продукту в каждом месяце прошлого года. При создании запросов этого типа можно воспользоваться Мастером перекрестных запросов (Crosstab Wizard).

- **Запрос на создание таблицы.** Когда вы превращаете запрос на выборку в запрос на создание таблицы, то записи, полученные в результате выполнения запроса, помещаются в новую таблицу.
- **Запрос на обновление.** С помощью запросов этого типа можно внести изменения в группу записей таблицы (например, в некотором поле изменить все прописные символы на строчные), а также выполнить редактирование данных, или внести изменения в одну таблицу, используя данные из другой.
- **Запрос на добавление.** Запросы этого типа позволяют добавлять данные из одной таблицы в другую.
- **Запрос на удаление.** Запрос на удаление позволяет исключить из таблицы целую группу записей, вместо того чтобы удалять каждую запись из таблицы вручную.

Как пользоваться каждым из перечисленных типов запросов, вы узнаете в следующем разделе «Составление сложных запросов».

Формы

Когда вы открываете в Access таблицу, данные в ней представлены в режиме таблицы.

В случае, когда записи в таблице содержат много полей и записей в таблице много, не всегда удобно просматривать данные непосредственно в таблице.

Именно по этой причине использование формы в большинстве случаев облегчает ввод, редактирование и просмотр данных.

Мастер Форм и Автоформы

Составление базы данных с нуля — задача довольно трудоемкая. В Access существует множество средств разработки, облегчающих процесс создания и настройки объектов. Одним из таких средств в Access 2000 является Мастер форм (Form Wizard). Мастер поможет вам создать форму в процессе диалога: в окне мастера необходимо ответить на несколько вопросов относительно макета и структуры формы. Форма, представленная в следующем разделе, была выполнена с помощью Мастера форм.

Если вас поджимает время, воспользуйтесь мастером Автоформ. Данный мастер создаст для вас один из перечисленных ниже типов форм. При запуске мастера Автоформ вам не потребуется давать никакой дополнительной информации, кроме названия таблицы или запроса, на основании которых будет построена форма.

Автоформа в столбец (Columnar). Тип формы с одной записью на страницу; все поля записи расположены столбцом.

Автоформа ленточная (Tabular). Тип формы с расположением полей строками, а не столбцами.

Автоформа табличная (Datasheet). Форма, в окне которой отображаются сразу несколько записей в том же виде, в котором данные представлены в таблице.

Специальные объекты в форме

Формы могут также содержать графику, гиперссылки на Web-страницы и документы, объекты **OLE** (объекты, связанные с другими приложениями Windows) и другие специальные объекты. Позже вы сможете выполнить несколько упражнений по созданию учебных форм и их элегантному оформлению в окне конструктора форм (Form Design).

Отчеты

Отчеты — еще один тип объектов в Access, который используется для просмотра и печати данных..

Вы можете создавать отчеты в Access с нуля. Но, по всей видимости, вам никогда не понадобится этого делать, учитывая наличие перечисленных ниже инструментов.

Мастер Отчетов (Report Wizard). Поможет вам поэтапно разработать отчет, начиная с выбора полей и заканчивая выбором стиля печатной страницы.

Автоотчет (AutoReport). С помощью Автоотчета можно одним щелчком мыши создавать ленточные отчеты и отчеты в столбец

Мастер Диаграмм (Chart Wizard). Поможет создавать различные графические объекты: от круговой диаграммы до трехмерной гистограммы, отображающей несколько наборов данных.

Мастер почтовых наклеек (Label Wizard). Поможет распечатать наклейки стандартного формата, а также документы с вашим собственным оформлением.

Создание новой базы данных

Создаем новую базу данных.. В главном окне Access нажмите кнопку Новая база данных. Откроется окно новой базы данных. Укажите имя файла базы данных, например, свою фамилию.

Окно базы данных

В каждой базе данных Access имеется окно базы данных. В этом окне находится панель Объекты (*Objects*) со следующими кнопками: Таблицы (*Tables*), Запросы (*Queries*), Формы (*Forms*), Отчеты (*Reports*), Страницы (*Pages*), Макросы (*Macros*) и Модули (*Modules*). В окне базы данных также имеется своя панель инструментов со следующими кнопками:

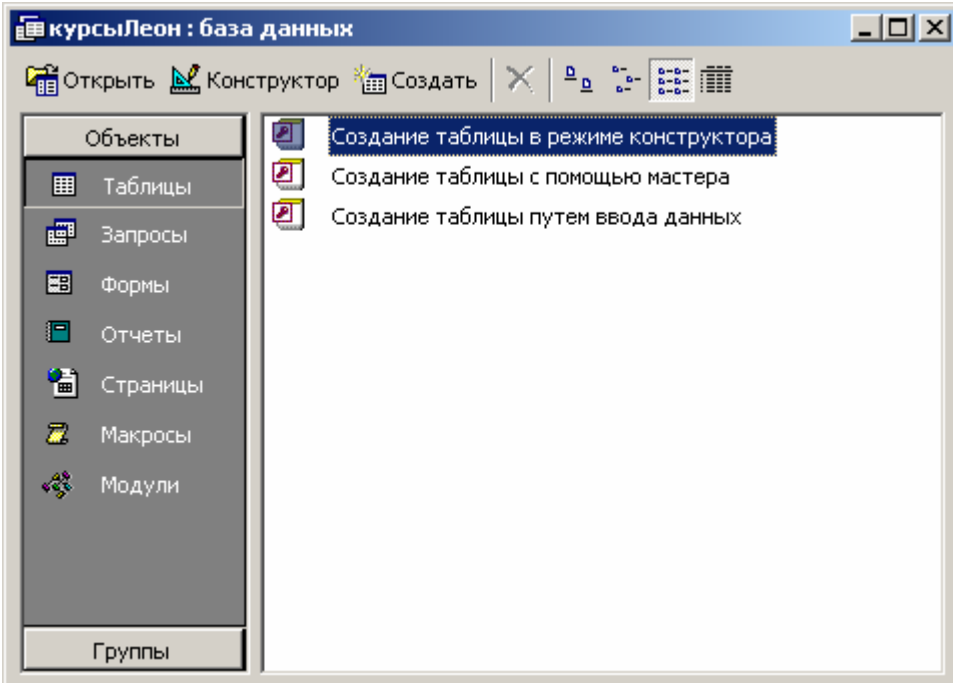


Рисунок 13.2. Окно базы данных Database1

- Открыть (Open) — для открытия объекта базы данных и работы с ним;
- Просмотр (Preview) — для открытия отчета в режиме предварительного просмотра (только для отчетов);
- Запуск (Run) — для запуска макроса или модуля (только для макросов и модулей);
- Конструктор (Design) — для изменения макета выбранного объекта;
- Создать (New) — для создания нового объекта базы данных.

Создание базы данных своими силами

При проектировании базы данных, прежде чем приступить непосредственно к созданию объектов — таблиц, форм и отчетов — следует посвятить некоторое время разработке детального плана всей базы данных в целом. При проектировании базы данных необходимо решить следующие вопросы:

- Выбор таблиц для базы данных;
- Выбор полей для таблиц;
- Способы ввода данных;
- Планирование форм, отчетов и запросов;

Выбор таблиц

При проектировании базы данных следует идти от общего к частному. Прежде чем углубляться в вопросы организации и структуры базы данных и перед тем, как перейти к разработке макетов форм и определения способов сортировки в отчетах (об этом мы будем говорить в последующих разделах данного урока), следует выяснить, какие необходимо создать таблицы. Последовательность проектирования базы данных:

- Соберите информацию относительно использования базы данных и проанализируйте механизмы работы с ней.
- Разделите информацию для базы данных по тематике (люди, проекты, заказы, товары для заказов, служащие и т. д.) и для каждой категории данных разработайте отдельную таблицу.
- Составьте список полей, которые необходимо будет включить в каждый тематический раздел.
- Удостоверьтесь, что поля в таблице не дублируются.

Сбор информации

Первый шаг в проектировании любой базы данных — это сбор информации с учетом дальнейшего использования базы данных. Вам следует провести анализ по следующим направлениям.

Какая информация войдет в базу данных.

Как будут обновляться данные, какая нужна обзорная и итоговая информация по данным и какие операции надо будет с ними выполнять.

Какие отчеты и диаграммы надо будет печатать и просматривать.

Но если вдруг вам повезло и у вас уже есть образцы форм и отчетов, вы можете их использовать в качестве отправной точки для своего плана.

Создадим новую базу данных Организация. База данных состоит из 3х таблиц: Отделы, Сотрудники и Проекты. Каждая таблица содержит несколько полей с разными типами данных.

Таблица 13.1. Таблицы и поля для реляционной базы данных Организация

Поля таблицы Отделы	Поля таблицы Сотрудник	Поля таблицы Проекты
Номер отдела	Номер сотрудника	Номер проекта
Название отдела	Имя	Название проекта
Этаж	Адрес	Стоимость
Начальник	Город	Дата начала
Адрес	Номер отдела	Дата окончания
	Номер проекта	Описание
	Зарплата	
	Дата рождения	
	Дата приема на работу	
	Должность	
	Образование	
	Телефон	

Представим данные в виде трех связанных таблиц

Связывание таблиц в Access

Чтобы отобразить данные одной таблицы совместно с относящимися к ним записями другой таблицы, в Access должно быть средство для связывания таблиц. Для существования связей между таблицами, таблицы должны иметь поля с общими значениями.

На этапе планирования базы данных убедитесь, что таблицы, данные которых необходимо будет использовать совместно, содержат общие поля.

Типы связей

Между двумя таблицами, если между ними установлена связь (как в предыдущем примере), существует некоторое отношение. В зависимости от того, могут ли значения в поле связи повторяться по несколько раз в той и другой связанной таблице, связь относится к одному из перечисленных ниже типов.

Один к одному (взаимно-однозначный). Этот тип связи существует, когда по обе стороны связи для любого значения в связующем поле имеется только одна запись. Например, между таблицей Заказы и таблицей Заказчики будет взаимно-однозначная связь, если они связаны по полю КодЗаказчика (у каждого заказа только один заказчик).

Один ко многим. Когда по одну сторону связи для каких-то значений в связанном поле может быть несколько записей, а по другую — только одна, то мы имеем дело со связью «один ко многим». Связь между таблицами Отделы и Сотрудники — пример

такой связи. Для каждого номера отдела в таблице Отделы имеется несколько записей в таблице Сотрудники.

Многие ко многим. Данный тип связи существует в случае, если значения в полях связи неоднократно встречаются в записях той и другой связанных таблиц.

Отсюда можно сделать вывод, что в Access тип связи между таблицами определяется одним фактором — являются или нет связанные поля полями ключа. Если таблица имеет ключевое поле (или ключ состоит из нескольких полей), это означает, что одно поле (или комбинация полей) было отведено под уникальный идентификатор для каждой записи таблицы. Каждая запись в таблице с ключевым полем должна иметь уникальное значение в поле ключа. Или, если полей ключа несколько, то комбинация полей ключа должна однозначно определять каждую запись таблицы. Благодаря этому ограничению Access «знает», какое количество записей допустимо для значений в некотором поле определенной таблицы. Например, если Access обращается к таблице Отделы, где ключевым является поле Номер отдела, то Access «знает», что в этой таблице для каждого значения в поле Номер отдела может иметься только одна запись.

Выяснить, какой тип связи существует между двумя таблицами, можно в окне Изменение связей (*Edit Relationships*). Щелкните правой кнопкой мыши на линии, соединяющей две таблицы, выберите команду Изменить связь (*Edit Relationship*) и задайте необходимые параметры в окне диалога Изменение связей (Edit Relationships). В этом окне вы можете выяснить, по какому типу связаны таблицы.

Выбор типа полей

Следующий этап после составления плана полей для таблиц — это выбор типа полей. Тип поля определяет данные, которые можно вводить в это поле, формат, который может иметь значения этого поля, и операции, которые можно выполнять с данными этого поля.

Типы полей в Access

В Access имеется множество типов полей. Они перечислены ниже с указанием назначения каждого типа.

- Счетчик (*AutoNumber*). В это поле Access автоматически вводит номер при добавлении каждой новой записи в таблицу.
- Текстовый (*Text*). Поле этого типа содержит текст: письма, числа и другие символы.
- Числовой (*Number*). В поле этого типа могут вводиться числовые данные любого формата.
- Дата (*DateTime*). Поле содержит дату и время в формате даты.
- Денежный (*Currency*). Этот тип поля предназначен для ввода денежных значений. Дата/время (*Date/Time*). В это поле вводится дата или время, либо их комбинация.
- Гиперссылка (*Hyperlink*). Данное поле содержит адреса гиперссылок, которые используются для перехода к Web-страницам, объектам базы данных или другим файлам.

- Мастер подстановок (*Lookup Wizard*). В поле этого типа запускается Мастер подстановок, который налагает ограничения на значения в поле. Значения/результаты поиска могут поступать из вводимого вами списка, таблицы или запроса.
- Поле MEMO (*Memo*). Поле этого типа может содержать текст неограниченной длины.
- Поле объекта OLE (*Object Linking and Embedding*, Связывание и внедрение объектов) содержит такие объекты, как рисунки и документы Word.
- Логический (*Yes/No*). В поле данного типа сохраняется одно из двух возможных значений: «истина» или «ложь». Этими значениями могут быть «да/нет», «истина/ложь», «мужчина/женщина» и т. д.

Ключевые поля

Ключ состоит из одного или нескольких полей, значения которых однозначно определяют каждую запись в таблице. Поля ключа обеспечивают выполнение некоторых особых функций базы данных.

Ключ необходим для связывания таблиц. Он также определяет природу связи. В таблице 2 показаны варианты связей между таблицами.

Access автоматически строит *первичный индекс* по полям ключа в таблице. Этот индекс упрощает поиск значений ключа и ускоряет любые поисковые и другие операции, которые используют значения в полях ключа.

На рис. 3 поле Номер отдела является полем ключа, или *первичным ключом* таблицы Отделы. В таблице Сотрудники это поле является *внешним ключом.* Так как для каждого значения в поле Номер отдела в таблице Отделы может быть только одна запись, а в таблице Сотрудники для того же самого значения — любое количество записей, то здесь мы имеем связь «один ко многим». Небольшие символы по обе стороны линии связи на рис. 3 показывают на тип связи.

Таблица 13.2. Влияние ключевых полей на тип связи таблиц

Связывающее поле является полем ключа	Тип связи
В обеих таблицах	Один к одному
В одной таблице	Один ко многим
Ни в одной таблице	Многие ко многим

Ввод данных

На следующем этапе планирования базы данных необходимо продумать процедуру ввода данных в таблицы. Очень важно, чтобы в процедуре ввода соблюдался принцип последовательности.

Связывание значений. Значения полей связи должны находиться под контролем. Например, не должно быть «висячих» записей: программа не установит связи между записями, в которых, к примеру, в одной таблице поле КодТовара содержит значение a1234, а в другой — A1234.

Группировка и сортировка записей. Если одинаковые значения полей, например имена, введены в разных записях по-разному, Access будет группировать и

сортировать записи не так, как бы вы того хотели. Так, запись Санкт-Петербург совершенно не соответствует записи С-Петербург.

Представление. Если в процедуре ввода данных нарушен принцип последовательности, то при отображении данных в форме или отчете могут возникнуть непоправимые ошибки.

Существуют приемы, позволяющие устранить неточности, часто возникающие при вводе данных в таблицу, но намного проще заранее определить условия на вводимые в поле значения. В Access существует несколько средств, позволяющих следить за вводом данных.

Определение значений по умолчанию

Если поле содержит, как правило (пусть не всегда), одно и то же значение, вы можете определить для такого поля значение по умолчанию. На этапе проектирования базы данных отметьте те поля, для которых можно было бы задать значения по умолчанию, — впоследствии это сократит время на ввод данных. Например, в таблице Хронометраж проектов вы могли бы определить для поля Описание значение по умолчанию Рабочие часы вне проекта. В качестве значений по умолчанию можно использовать функции Visual Basic. Если вы установите в качестве значения по умолчанию функцию Date( ), то при добавлении каждой новой записи в это поле будет автоматически вводиться текущая дата.

Проверка данных по маске ввода

Маски ввода (InputMask) обеспечивают два различных вида управления вводом данных. Они применяются в следующих целях:

- для проверки значения поля на соответствие определенному шаблону ввода, например, шаблон 000-00-0000 задает ввод номера социального обеспечения; |
- для автоматического ввода постоянных символов-разделителей, например, дефиса (-) или круглых скобок (());
- для обозначения позиций ввода с тем, чтобы облегчить ввод данных;
- для сохранения данных вместе с константами или без констант, в зависимости от вашего предпочтения.

Создание маски ввода для поля облегчено тем, что существует специальный мастер, помогающий настраивать существующие маски и создавать новые.

Режим конструктора таблиц

Окно *конструктора таблиц* — средство Access для работы с макетом таблицы. В нем самостоятельно выбирается тип данных для каждого поля, а также такие свойства полей, как значения по умолчанию, маски ввода и поля подстановок.

Независимо оттого, каким методом вы пользуетесь при создании таблиц, в дальнейшем всегда можно открыть таблицу в режиме конструктора, просмотреть ее макет и внести необходимые изменения.

Режим таблицы

Когда вы открываете в Access таблицу, она загружается в окно, в котором данные представлены в виде строк (записей) и столбцов (полей). Такой режим представления

данных называется *режимом таблицы*. В Access существует средство для непосредственного ввода данных в новую таблицу, открытую в режиме таблицы.

Процедура создания таблицы в режиме таблицы выглядит следующим образом.

1. В окне базы данных щелкните на кнопке Таблицы (Tables).
2. Дважды щелкните на строке Создание таблицы путем ввода данных (Create Table By Entering Data). (Вы можете также щелкнуть на кнопке Создать (New), а затем в диалоговом окне Новая таблица (New Table) дважды щелкнуть на пункте Режим таблицы (Datasheet View).)
3. Access откроет в режиме таблицы пустой бланк.

Работа с конструктором таблиц

В окне конструктора таблиц можно создать новую таблицу или изменить макет уже существующей.

Создание новой таблицы в режиме конструктора

Чтобы создать новую таблицу в режиме конструктора, выполните следующую процедуру.

1.Щелкните на кнопке **Таблицы** (Tables) в окне базы данных.

2. Дважды щелкните на строке **Создание таблицы в режиме конструктора** (Create TableIn Design View) (или щелкните на кнопке Создать (New), а затем — дважды напункте Конструктор (Design View) в диалоговом окне **Новая таблица** (New Table)).Откроется окно конструктора таблиц с незаполненным бланком таблицы (рис. 3.6.).

3. Добавьте поля и задайте их свойства.

4. Закройте окно конструктора. В ответ на предложение сохранить изменения щелкните на кнопке **Да** (Yes), введите название для новой таблицы и щелкните на кнопке **ОК**.

5. Если Access выдает запрос с предложением определить ключ, щелкните на кнопке **Да** (Yes), чтобы добавить в таблицу ключевое поле типа Счетчик (AutoNumber).Если вы надеетесь позже сами определить ключевое поле или оставить таблицу без ключа, щелкните на кнопке **Нет** (No). В итоге вы вернетесь в окно базы данных, а в списке объектов окна появится новая таблица.

Добавление поля

Сначала в пустой бланк конструктора таблиц следует добавить поля.

1. В первую строку столбца **Имя поля** (Field Name) введите имя поля, которое не может быть длиннее 64 символов и может включать буквы, числа, пробелы и другие символы, кроме символов точки (.), восклицательного знака (!), надстрочного символа (") и прямых скобок ([]). Сведения о других ограничениях на имена полей вы найдете в Справочной системе Access.
2. Перейдите к столбцу **Тип данных** (Data Type) и выберите тип данных для поля. Вы можете ввести тип поля самостоятельно или выбрать его в раскрывающемся списке типов данных. (Стрелка этого раскрывающегося

списка становится видимой после того, как курсор окажется в ячейке столбца Тип данных (DataType).) Типы данных Access описаны в разделе, посвященном выбору полей для таблицы.

3. При выборе типа **Текстовый** (Text) или **Числовой** (Number) у поля появляется свойство **Размер поля** (Field Size). Длина текстовых полей по умолчанию — 50 символов числовых — длинное целое. Чтобы изменить это свойство, убедитесь, что в нижней части окна конструктора раскрыта вкладка **Общие** (General). Далее для текстового поля щелкните на строке **Размер поля** (Field Size) и измените значение. Для числового поля щелкните на строке **Размер поля** (Field Size) и выберите значение в раскрывающемся списке, который состоит из значений: Байт (Byte), Целое (Integer), Длинное целое (Long), С плавающей точкой (4 байта) (Single), С плавающей точкой (8 байт) (Double) или Код репликации (Replication ID)
4. Чтобы добавить следующее поле, переместите курсор на следующую строку в столбце **Имя поля** (Field Name) и выполните шаги 1-3.
5. Чтобы для текстового и числового полей изменить значение размера поля по умолчанию, выберите команду **Сервис**(*Tools*) > **Параметры** (*Options*). В диалоговом окне Параметры (*Options*) перейдите на вкладку Таблицы и запросы (*Tables/Queries*) и измените размеры полей по умолчанию.

Добавление ключевых полей

Кключ таблицы должен иметь уникальное значение для каждой записи в таблице. Ключ может состоять из нескольких полей. Он необходим для формирования первичного индекса таблицы. Первичный индекс ускоряет поиск по ключевому полю (полям) и используется для определения типа связи между таблицами при объединении таблиц (например, «один к одному» или «один ко многим»).

После беглого обзора назначения и функций ключа давайте сразу же перейдем к практике и добавим в таблицу ключ.

1. Щелкните на области выделения поля, которое вы хотите сделать ключевым. Для выделения смежных полей щелкните на первом из них и, не отпуская кнопки мыши, протащите указатель по остальным полям. Для выделения нескольких несмежных полей нажмите клавишу Ctrl и, не отпуская ее, щелкайте на необходимых полях.)
2. Щелкните на кнопке Ключевое поле (*Primary Key*) на панели инструментов или выберите пункт Ключевое поле (*Primary Key*) в меню Правка (Edit).
Ключевое поле помечается значком ключа в области выделения строки

Удаление ключевого поля

Поле (поля) ключа удаляется следующим образом.

1. Выделите поле (поля) ключа щелчком на области выделения строки, в которой находится поле.

2. Щелкните на кнопке Ключевое поле (Primary Key) на панели инструментов или выберите пункт Ключевое поле (Primary Key) в меню Правка (Edit).
Значок ключа в области (областях) выделения строки (строк) исчезнет. Кнопка Ключевое поле (Primary Key) на панели инструментов, как и пункт Ключевое поле! (Primary Key) в меню Правка (Edit), действуют как флажки.

Определение отношений между таблицами

В предыдущем разделе, посвященном проектированию баз данных, вы узнали, что такое отношения, или связи таблиц, и как Access их использует для объединения таблиц. Вы также узнали о типах связей, существующих между таблицами («один к одному», «один ко многим», «многие ко многим») и о том, как они определяются наличием или отсутствием ключа в полях связи.

Чтобы открыть окно Схем данных (Relationships), выберите в меню Сервис (Tools) пункт Схема данных (Relationships) или щелкните правой кнопкой мыши в любом месте окна базы данных и выберите пункт Схема данных (Relationships) в контекстном меню.

На рисунке 13.3 **Схема данных** (Relationships) можно видеть, как связаны таблицы, а также устанавливать связи, изменять параметры целостности и определять типы объединения

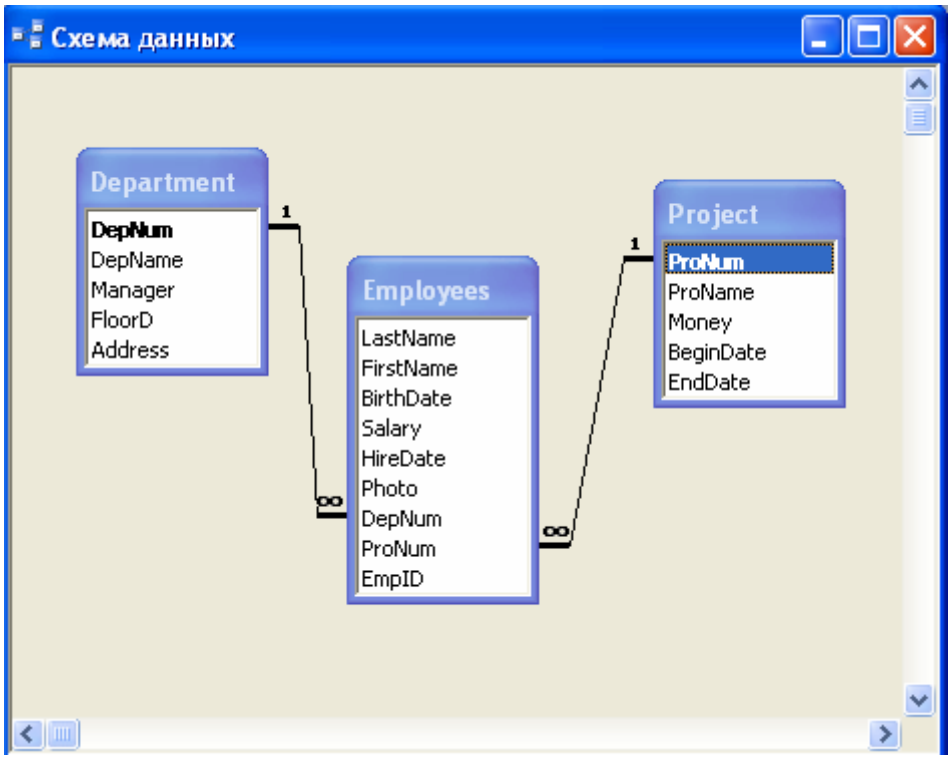


Рисунок 13.3. Схема базы данных

Установление связи

Чтобы самостоятельно установить связь в окне Схема данных (*Relationships*):

1. Откройте окно **Схема данных** (*Relationships*), как описано в предыдущем разделе.
2. Щелкните на кнопке **Отобразить таблицу** (*Show Table*) на панели инструментов.
3. В диалоговом окне **Добавление таблицы** (*Show Table*) двойным щелчком на имени таблицы по очереди выберите необходимые таблицы, если они не отображаются на схеме данных.
4. После того как все необходимые таблицы будут добавлены в окно схемы данных, щелкните на кнопке **Закрыть** (*Close*) в диалоговом окне **Добавление** таблицы (*Show Table*).
5. Нажмите кнопку мыши на поле связи одной таблицы и перетащите его на поле связи в другую таблицу. Как только вы отпустите кнопку мыши, откроется диалоговое окно **Изменение связей** (*Edit Relationships*):
6. Если необходимо связать таблицы по нескольким полям, щелкните на второй строке в столбце **Таблица/запрос** (*Table/Query*) и выберите в раскрывающемся списке следующее поле связи. В списке второй строки столбца **Связанная таблица/запрос** (*Related Table/Query*) выберите поле связи другой таблицы. Повторите этот шаг для каждой дополнительной пары полей, по которым следует связать таблицы.
7. Щелкните на кнопке **Создать** (*Create*), чтобы установить связь. В окне **Схема данных** (*Relationships*) появится линия новой связи, как показано на рис. 3.

Удаление связи

Чтобы удалить связь, выполните следующие действия.

1. В окне Схема данных (Relationships) щелкните правой кнопкой мыши на линии связи, которую вы хотите удалить
2. В контекстном меню выберите команду Удалить (Delete).
3. Щелкните на кнопке Да (Yes), чтобы подтвердить удаление.

Если вы не находите в контекстном меню команды Удалить (Delete), щелкните сначала на линии связи, чтобы ее выделить. Затем снова щелкните на линии связи правой кнопкой мыши.

Сохранение изменений схемы данных

Чтобы сохранить результаты своей работы в окне Схема данных (*Relationships*),достаточно закрыть окно и в появившемся диалоговом окне запроса щелкнуть на кнопке Да (*Yes*).

Объекты OLE

Поля типа **Поле объекта OLE *(OLE Object Linking and Embedding)*** содержат такие данные, как рисунки, документы Word, звукозаписи или объекты иного формата, созданные в других приложениях. Когда вы вводите данные в поле объекта OLE, у вас есть два варианта. Вы можете создать:

- связанный объект, который содержит ссылку на рисунок, документ или другой объект OLE в таблице Access. При двойном щелчке на поле объекта OLE, Windows открывает приложение, в котором был создан объект, и вы можете его t изменить;
- внедренный объект, который содержит копию объекта OLE в таблице. Изменения, вносимые в исходный объект, не отражаются на объекте, хранящемся в базе данных.

Вставка связанного объекта OLE

Поле **Фото** в таблице **Сотрудники** имеет тип **Поле объекта OLE (OLE Objec**t). Чтобы понять, как работает объект OLE, попробуйте вставить в это поле связанное изображение (уже существующий файл с расширением .jpg).

1. В режиме таблицы установите курсор в поле **Фото** той записи, которую вы редактируете.
2. Щелкните правой кнопкой на поле и выберите в контекстном меню команду **Добавить объект** (Insert Object) либо выберите команду **Вставка > Объект** (Insert» •Object). Откроется показанное ниже диалоговое окно **Вставка объекта** (Insert Object). Состав списка **Тип объекта** (Object Type) зависит от программ, установленных на вашем компьютере.
3. Установите переключатель **Создать из файла** (Create from File).
4. После того как изменится вид диалогового окна, щелкните на кнопке **Обзор** (Browse).
5. В диалоговом окне Обзор (Browse) выберите для связи файл с рисунком. Например, файл C:\Мои документы\Мои рисунки\нпд.jpg.
6. Вернувшись в диалоговое окно **Вставка объекта** (Insert Object), установите флажок Связь (Link).
7. Если вы предпочитаете, чтобы в поле объекта OLE объект отображался в виде значка, установите флажок **В виде значка** (Display as Icon). В результате вместо названия типа объекта в поле будет отображен его значок.
8. Щелкните на кнопке **ОК**, чтобы вернуться к таблице в режиме таблицы.

В окне в режиме таблицы вы увидите ссылку на тип связанного объекта OLE. Например, если файлы с расширением .jpg связаны на вашем компьютере с приложением Microsoft Photo Editor, тогда поле объекта OLE будет выглядеть следующим образом:

Фотография Microsoft Photo Editor 3.0

Вставка внедренного объекта OLE

Если вы хотите хранить копию фотографии в таблице Сотрудники (а не связывать объект в таблице с исходным файлом), необходимо добавить в поле Фото внедренный объект.

1. Перейдите в поле **Фото** (или любое другое поле объекта OLE, с которым вы работаете).
2. Щелкните правой кнопкой в ячейке поля и выберите в контекстном меню команду **Добавить объект** (Insert Object).
3. Установите переключатель **Создать из файла** (Create from File).
4. Щелкните на кнопке **Обзор** (Browse) и выберите файл рисунка.

5. Щелкните на кнопке ОК в диалоговом окне **Вставка объекта** (Insert Object), не устанавливая флажка **Связь** (Link).
6. Вернувшись к окну таблицы, вы увидите в поле объекта OLE такой же значок (или указание на тип объекта), как в примере из предыдущего раздела. Вы можете открыть объект в его «родном» приложении и отредактировать его. Но при этом все изменения сохранятся только в копии объекта, хранящейся в поле объекта OLE, — в исходном файле изменения не отразятся.

Просмотр объекта OLE

Чтобы просмотреть содержимое поля объекта OLE, дважды щелкните на нем, чтобы открыть приложение, в котором был создан объект. Например, если вы вставили связанный документ Word, то при двойном щелчке на этом объекте документ откроется в окне приложения Word, где вы сможете его отредактировать.

Редактирование объекта OLE

Чтобы отредактировать объект OLE, откройте его двойным щелчком в исходном приложении, как описано выше. В окне приложения вам доступны все средства данного редактора. При сохранении изменений по окончании редактирования все они отразятся в объекте, и, открыв объект вновь, вы увидите его в последней редакции.

Основное различие между редактированием связанных и внедренных объектов заключается в следующем. Любые изменения, которые вы вносите в связанный объект, непосредственно передаются в исходный объект. В случае с внедренным объектом все изменения отражаются только на копии объекта, которая хранится в Access, — исходный объект при этом не изменяется.

Автоформы

Самый быстрый способ создать форму для таблицы или запроса — построить ее с помощью Автоформы. *Автоформа* — это средство мгновенного создания форм без выдачи дополнительных запросов, в отличие от Мастера форм, который ведете пользователем диалог. Существует три разновидности Автоформ:

- Автоформа в столбец;
- ленточная Автоформа;
- табличная Автоформа.

Перед тем как освоить несложную процедуру создания форм с помощью Автоформ, посмотрим, как они выглядят.

Автоформа в столбец

Автоформа в столбец отображает в режиме формы только одну запись.

Картинка в поле Фото не занимает своего должного места в рамке. Это как раз тот случай, когда необходимо уметь отредактировать базовую форму, созданную Access с помощью Автоформы.

Ленточная Автоформа

В ленточной Автоформе одновременно отображается несколько записей, поля которых располагаются по строкам.

Табличная Автоформа

В табличной Автоформе записи отображаются так же, как в режиме таблицы.

Может возникнуть вопрос: зачем нужна табличная Автоформа. Во-первых, табличная Автоформа понадобится нам для отображения связанных записей в форме на базе нескольких таблиц.

Существует и другое применение табличной Автоформы. В Автоформу (или любую другую форму) в режиме формы при необходимости можно добавить графику и кнопки. В режиме таблицы это сделать невозможно.

Создание Автоформы

В этом разделе описаны процедуры создания различных видов Автоформ, включая два вида форм, отображающих данные из нескольких таблиц.

Создание Автоформы в столбец

Процедура создания Автоформы в столбец является наиболее простой и состоит из двух шагов.

1. В окне базы данных щелкните на кнопке **Таблицы** (Tables) (или на кнопке **Запросы** (Queries)), а затем — на имени объекта, для которого необходимо создать Автоформу.
2. В раскрывающемся списке кнопки **Новый объект** (New Object) на панели инструментов **База данных** (Database) выберите команду **Автоформа** (AutoForm).

Итог —Автоформа наподобие той, что показана на рисунке

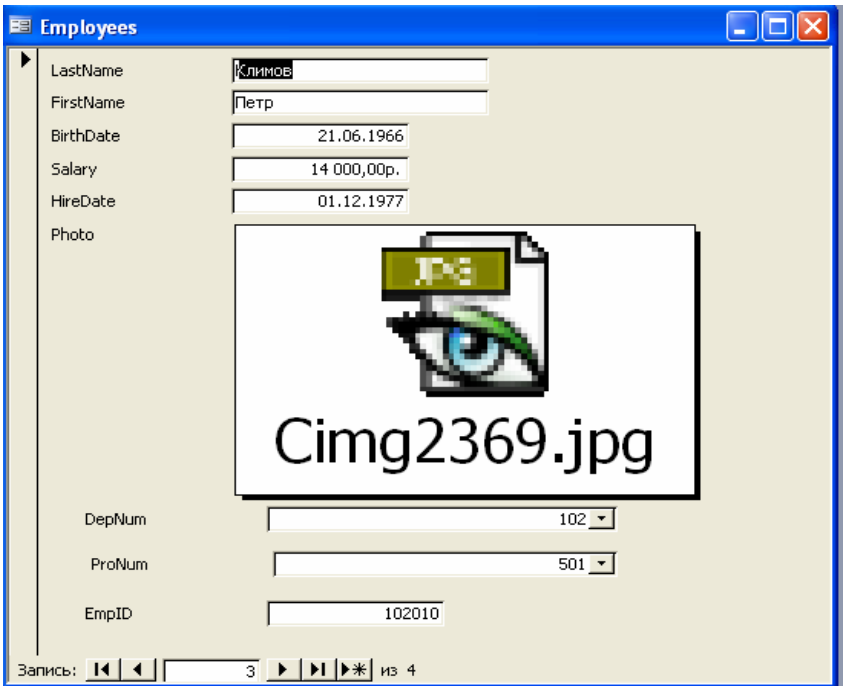


Рисунок 13.4. Пример автоформы

Окно конструктора форм

Конструктор форм представляет собой окно редактора, в котором можно вносить изменения в макет формы. В данном окне доступны следующие инструменты проектирования:

- панель инструментов Конструктор форм (Form Design);
- панель инструментов Формат (форма/отчет) (Formatting (Form/Report));
- панель инструментов Панель элементов (Toolbox);
- список полей;
- таблица свойств;
- мастеры по созданию различных элементов управления.

Перед тем как приступить к работе над формами, давайте рассмотрим каждый из перечисленных компонентов. Для начала давайте откроем окно конструктора форм

Открытие формы в режиме конструктора

Способ открытия окна конструктора зависит от ситуации.

- В процессе создания формы с помощью Мастера форм перейти в режим конструктора можно, указав на последнем шаге мастера, что вы собираетесь внести изменения в макет формы.
- В окне базы данных необходимо щелкнуть на кнопке Формы (Forms), выделить имя формы и щелкнуть на кнопке Конструктор (Design) либо выделить имя формы щелчком правой кнопки и в контекстном меню выбрать команду Конструктор (Design).
- Если вы просматриваете форму, щелкните на кнопке Вид (View) на панели инструментов.

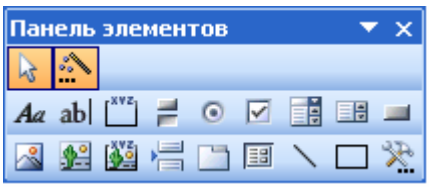
Панель инструментов Формат

Панель инструментов Формат (форма/отчет) (Formatting (Form/Report)) появляется как в режиме конструктора форм, так и в режиме конструктора отчетов. Большая часть кнопок этой панели выполняет такие стандартные функции форматирования, как изменение размеров и цвета шрифта. В этом разделе мы не будем подробно останавливаться на том, как пользоваться этими кнопками (работа с ними не представляет никаких трудностей). Кроме того, в примерах данного урока вы выполните действия, связанные с использованием этих кнопок.

Первый специальный инструмент панели инструментов Формат (форма/отчет) (Formatting (Form/Report)), на котором следовало бы остановиться, — это раскрывающийся список Элемент управления (Object). Этот список позволяет выбирать элементы управления (такие как поле или надпись).

Панель элементов

Па панели элементов расположены кнопки для размещения в форме элементов управления, таких как Надпись (Label), Поле (Text Box), Поле со списком (Combo Box), Кнопка (Command Button) и других особых элементов управления формы. Чтобы вывести панель элементов на экран, щелкните на одноименной кнопке на панели инструментов Конструктор форм (Form Design).



Список полей

Список полей — это удобное средство связывания элемента управления в форме с определенным полем таблицы или запроса. Например, чтобы создать в форме элемент управления для поля Фамилия, следует щелкнуть на кнопке Поле (Text Box) на панели элементов, а затем из списка полей перетащить поле Фамилия в форму.

Чтобы вывести список полей на экран, щелкните на одноименной кнопке панели инструментов Конструктор форм (Form Design). Список полей отображает поля таблицы (или запроса), являющейся источником данных для формы, с которой вы работаете.

Добавление в форму полей

Если вы добавляете в таблицу поле после того, как созданы формы, то это поле не войдет ни в одну из существующих форм. В этом случае вам придется вернуться к нужной форме и добавить поле вручную. Выполнить эту задачу можно различными способами. Следующий пример демонстрирует, как можно внести поле Телефон в форму Сотрудники.

Добавление поля с помощью списка полей

Чтобы добавить в форму поле из списка полей, выполните описанную ниже процедуру.

- Щелкните на кнопке Список полей (Field List) на панели инструментов Конструктор форм (Form Design).
- Перетащите поле Телефон из списка полей в форму.
- При необходимости поменяйте положение нового элемента управления.

Запросы и фильтры

Для поиска и отбора записей используются как фильтры, так и запросы, но у каждого из этих двух средств есть свои особенности по настройке и применению. На основании этих особенностей можно сформулировать два кратких правила:

- фильтр следует использовать, если из набора записей, которые вы в текущий момент просматриваете в таблице или форме, необходимо оперативно сделать выборку в соответствии с определенными критериями;
- запрос предназначен для выборки и просмотра записей из одной или нескольких таблиц, его можно сохранить, чтобы позднее вновь использовать или создать на его базе форму или отчет.

В таблице дана более подробная сравнительная характеристика фильтра и запроса.

Вычисляемые поля

Во многих случаях в таблицах хранятся не только данные, но и программы их обработки.

В Access программы обработки данных пишутся на Visual Basic for Applications (VBA)

По причине ограниченности издания мы приведем только один пример, иллюстрирующий использование VBA.

Пусть необходимо для изделий, хранимых на складе, автоматически пересчитывать и сохранять в базе данных стоимость остатка при изменении цены изделия или текущего остатка.

Информация об изделиях хранится в таблице IZD — справочнике изделий, содержащей сведения: о цене — в поле PRICE, о текущем остатке — в поле TEK_OST и его стоимости в поле STOIM. Для просмотра и корректировок справочника на основе таблицы IZD создана форма (рис. 13.5.). Для того чтобы при изменении через форму значений цены или текущего остатка изделия новое расчетное значение стоимости остатка заносилось в таблицу, создадим процедуру обработки события AfterUpdate (после обновления) для полей PRICE и STOIM

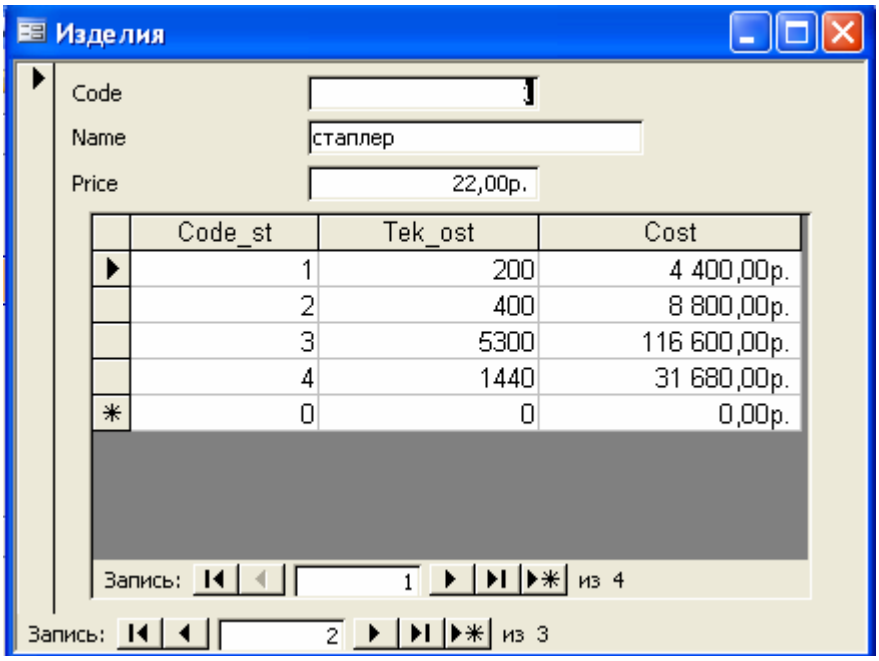


Рис13.5 Форма для работы со справочником изделий

Для этого откроем форму в режиме конструктора и окно свойств поля цены PRICE. На вкладке **События** в строке **После обновления** (AfterUpdate) выберем значение **[Процедура обработки события].** В открывшемся окне модуля формы (рис. 13.6.) в шаблон процедуры обработки события Private sub Price_AfterUpdate( ) для поля PRICE запишем инструкцию присваивания:

```
Option Compare Database
Private Sub Тек_остаток_AfterUpdate()
Стоимость = Цена * Тек_остаток
End Sub

Private Sub Цена_AfterUpdate()
Стоимость = Цена * Тек_остаток
End Sub
```

Рис 13.6.. Окно модуля формы

Эта инструкция заносит в поле STOIM новое рассчитываемое значение. Такую же инструкцию запишем в шаблон процедуры Private sub Tek_ost afterUpdate( ) для поля Tek_ost

Событие ***AfterUpdate*** (после обновления) для элемента управления "поле" возникает после того, как пользователь ввел новое или изменил существующее значение в поле и перешел к другому элементу управления или выполнил команду **Сохранить запись** меню **Записи.**

Пусть при изменении цены изделия, хранимого на нескольких складах, необходимо автоматически пересчитывать стоимость текущих остатков изделия на каждом складе.

Справочная информация об изделиях хранится в таблице ИЗДЕЛИЯ, содержащей поля КОД (код изделия) и ЦЕНА. Данные о текущих остатках изделий на складах и их стоимости хранятся в подчиненной таблице ОСТАТОК с полями КОД (код изделия), КОД_СКЛАДА, ТЕК_ОСТ и СТОИМОСТЬ.

Для внесения изменений в справочную информацию об изделии и его цене используется форма, созданная на основе таблицы ИЗДЕЛИЯ (рис. 13.7.).

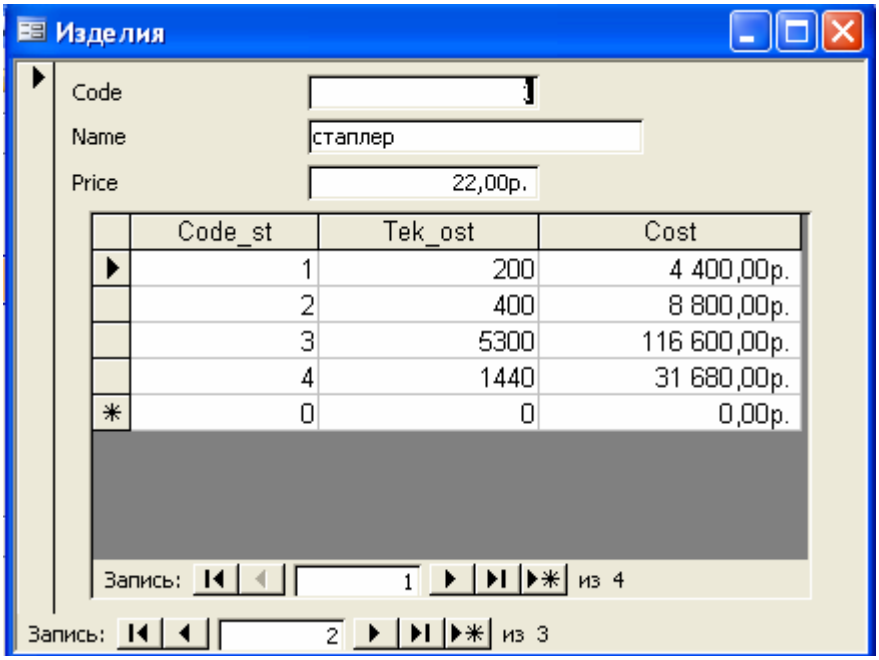


Рис. 13.7. Форма для работы со справочником изделий

После изменения цены должен автоматически происходить перерасчет стоимости остатков на каждом складе, где имеется это изделие. В результате перерасчета должно обновляться поле СТОИМОСТЬ в таблице ОСТАТОК.

Для просмотра результатов перерасчета в таблице ОСТАТОК используем одноименную форму, приведенную на рис. 13.8.

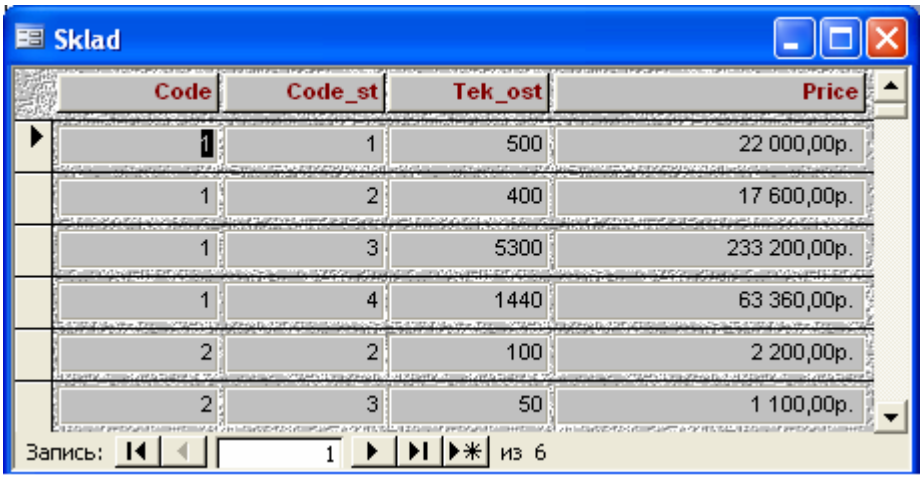


Рис. 13.8. Форма для просмотра текущих остатков изделий на складах и их стоимости

Для того чтобы при изменении через форму ИЗДЕЛИЯ значений цены рассчитывались новью значения стоимости остатков на складах и заносились в таблицу ОСТАТОК, создадим процедуру обработки события Afterupdate для поля ЦЕНА. Для этого откроем форму ИЗДЕЛИЯ в режиме конструктора и окно свойств поля ЦЕНА. На вкладке **События** в строке **После обновления** выберем значение **[Процедура обработки события].** В открывшемся окне модуля формы (см. рис. 7.52) в шаблон процедуры обработки события Private Sub UEHA_AfterUpdate () для поля ЦЕНА запишем инструкции:

```
Option Compare Database
Option Explicit

Private Sub Code_AfterUpdate()

End Sub

Private Sub Price_AfterUpdate()
On Error GoTo Err_Knopka2_Click
Dim dbs As Database
Dim stab As Recordset
Dim Col As Integer
```

```
    Dim stDocName As String
    Dim stLinkCriteria As String
    Set dbs = CurrentDb()
    Set stab = dbs.OpenRecordset("SELECT SKLAD.* from sklad inner join
èçäåëèÿ on sklad.code = èçäåëèÿ.code")
    MsgBox "New Price " & Price & " for & code " & Code, vbOKOnly
    Col = 0
    Do Until stab.EOF
    stab.Edit
    If stab!Code = Code Then
    Col = Col + 1
    stab!Cost = Price * stab!Tek_ost
    stab.Update
    End If
    stab.MoveNext
    Loop
    MsgBox "Updated " & Col & " Records", vbOKOnly
    stDocName = "Sklad"
    stLinkCriteria = "[Code]=" & "'" & Me![Code] & "'"
    DoCmd.OpenForm stDocName, , , stLinkCriteria
    Exit_Knopka2_Click:
    Exit Sub

    Err_Knopka2_Click:
    MsgBox Err.Description
    Resume Exit_Knopka2_Click
    End Sub
```

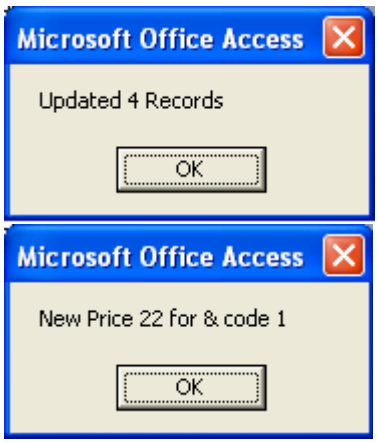


Рис. 13.9. Сообщения процедуры ЦЕНА Af terUpdate ()

Создание отчета

В Access входит несколько средств для создания отчетов:
Автоотчет;

Мастер отчетов (для создания отчетов, основанных на одной или нескольких-таблицах);

Мастер почтовых наклеек;

Мастер диаграмм;

Конструктор отчетов.

Автоотчет

Автоотчет (AutoReport) — это наиболее простое средство Access для создания отчета на основе таблицы или запроса. Это средство работает быстрее, чем мастер. Все что необходимо сделать для создания отчета с помощью этого средства указать имя таблицы или запроса и при желании выбрать тип Автоотчета: ленточный или в столбец. Access включает в отчет все поля базовой таблицы или запроса,

Автоотчет в столбец

Автоотчет в столбец (Columnar AutoReport) отображает записи из таблицы и запроса и располагает эти записи по столбцам.

Создание Автоотчета в столбец с применением последнего использованного стиля

При создании Автоотчета с помощью диалогового окна Новый отчет (New Report) ему назначается стиль, который назначался при последнем запуске Мастера отчетов или при последнем использовании Автоформата. Если ни одного из этих средств создания отчетов еще не было задействовано, Автоотчету назначается стиль Обычный (Normal).

Чтобы создать Автоотчет с применением последнего использованного стиля, выполните следующие действия.

Щелкните на кнопке Отчеты (Reports) в окне базы данных.

Щелкните на кнопке Создать (Normal). Откроется диалоговое окно Новый отчет (New Report).

В нижней части диалогового окна Новый отчет (New Report) выберите в раскрывающемся списке таблицу или запрос для отчета.

Щелкните дважды на пункте Автоотчет: в столбец (AutoReport: Columnar).

Когда вы закроете окно отчета, в ответ на предложение сохранить изменении щелкните на кнопке Да (Yes) и задайте имя для нового отчета.

Ленточный автоотчет

Ленточный автоотчет (Tabular Auto Report) отображает записи в виде таблицы. При таком оформлении каждое поле базовой таблицы или запроса представлено отдельным столбцом, а каждая запись — отдельной строкой. В результате данные отображаются точно так же, как в таблице или запросе в режиме

Для создания ленточного Автоотчета выполните следующие действия.

В окне базы данных щелкните на кнопке Отчеты (Reports)

Щелкните на кнопке Создать (New). Откроется диалоговое окно Новыйотчет (New Report).

В нижней части диалогового окна Новый отчет (New Report) выберите в раскрывающемся списке таблицу или запрос для отчета.

Щелкните дважды на пункте Автоотчет: ленточный (AutoReport: Tabular).

Когда вы закроете окно отчета, в ответ на предложение сохранить изменения щелкните на кнопке Да (Yes) и задайте имя для нового отчета.

Главная кнопочная форма

После составления списка основных задач можно выделить категории, на основании которых будут создаваться кнопки для главной кнопочной формы приложения. Перед тем как двигаться дальше, имеет смысл освежить ваши знания по этой теме. В приложении Хронология мы выделяем следующие категории:

- еженедельное ведение записей;
- печать отчетов;
- обновление данных о сотрудниках.

В процессе разработки кнопочной формы в список всегда можно добавить новые категории.

Создание кнопочных форм

Когда план разработки приложения составлен, можно приступать к созданию кнопочных форм. *Кнопочная форма (Switchboard)* — это всего-навсего обыкновенная форма с кнопками, обеспечивающими возможность открытия других форм (обычных или кнопочных), а также выполнение таких операций, как печать отчетов. В Access существует специальное средство, облегчающее создание кнопочных форм. Диспетчер кнопочных форм (Switchboard Manager) поможет в создании кнопочных форм и в дальнейшей их модификации по мере изменения самой базы данных.

Чтобы запустить Диспетчер кнопочных форм, выполните следующую процедуру.

Выберите команду Сервис > Служебные программы *I* Диспетчер кнопочных форм
(Tools > Database > Utilities > Switchboard Manager).

В ответ на предложение создать кнопочную форму щелкните на кнопке Да (Yes). Откроется диалоговое окно.

Каждая кнопочная форма в приложении называется *страницей главной кнопочной формы (switchboardpage)*. Когда на шаге 2 предыдущей процедуры вы щелкаете на кнопке Да (Yes), диспетчер добавляет в диалоговое окно страницу Main Switchboard (Default) (главная кнопочная форма по умолчанию). Чтобы перед добавлением кнопок увидеть эту кнопочную форму, выполните следующие действия.

1. Закройте диалоговое окно Диспетчер кнопочных форм (Switchboard Manager).
2. В окне базы данных щелкните на кнопке Формы (Forms).
3. Выберите форму Switchboard и щелкните на кнопке Открыть (Open) (или просто
дважды щелкните на форме).

Добавление элементов в главную кнопочную форму

Добавим в главную кнопочную форму базы данных необходимые элементы.

1. Если в окне Диспетчер кнопочных форм (Switchboard Manager) имеется несколько страниц, выделите главную страницу Main Switchboard (Default).
2. Щелкните на кнопке Изменить (Edit). Откроется следующее диалоговое окно:
3. Чтобы добавить элемент, щелкните на кнопке Создать (New). Откроется диалоговое окно Изменение элемента кнопочной формы (Edit Switchboard Item):
4. Введите в поле Текст (Text) имя нового элемента Еженедельник.
5. В раскрывающемся списке Команда (Command) выберите команду Открыть форму для изменения (Open Form In Edit Mode).
6. В раскрывающемся списке Форма (Form) выберите форму Еженедельник.
7. Щелкните на кнопке О К, чтобы вернуться к предыдущему окну.
8. Повторите шаги 3-7 для каждого элемента кнопочной формы.

Размещение элементов на кнопочной форме

После того как элементы страницы созданы, можно разместить их требуемым образом на странице. Прежде всего, в окне Диспетчер кнопочных форм (Switchboard Manager) выберите страницу, которую необходимо отредактировать. Затем щелкните на кнопке Изменить (Edit) и в диалоговом окне Изменение элемента кнопочной формы (Edit Switchboard Item) выполните следующие действия:

- чтобы переместить элемент страницы вверх, выделите элемент и щелкните на кнопке Вверх (Move Up) столько раз, сколько необходимо, чтобы он занял нужную позицию;
- чтобы переместить элемент страницы вниз, выделите элемент и щелкните на кнопке Вниз (Move Down) столько раз, сколько необходимо, чтобы он занял нужную позицию.

По окончании работы с элементами страницы щелкните на кнопке Закрыть (Close), чтобы вернуться к окну Диспетчер кнопочных форм (Switchboard Manager).

Работа Access с Microsoft SQLServer в сети .

Выполнение настройки совместной работы SQL Server и Access в сети можно условно разбить на 4 этапа:

1. Создание базы данных на SQL Server
2. Установка связи между Access и SQL Server (создание источника данных)
3. Создание запроса к серверу в среде Access
4. Запуск запроса из среды Access

Установка связи между Access и SQL Server.

Access и SQL Server работают совместно, используя интерфейс ODBC. При помощи этого интерфейса создаются так называемые «источники данных». При

запуске запроса к серверу из Access пользователю необходимо выбрать один из источников данных.

Теперь рассмотрим как создать источник данных, то есть связать между собой базу данных SQL Server, которая располагается на сервере и запрос Access с клиента сети. Изложенную ниже последовательность действий необходимо выполнить на **компьютере-клиенте**:

Необходимо запустить «Администратор ODBC»: (**Мой компьютер - Панель управления - Источники данных ODBC**)

Далее необходимо выбрать закладку **«Файловый DSN»** и нажать кнопку **Добавить.** После этого запускается мастер создания источника данных.

В первом окне мастера необходимо в списке выбрать драйвер источника данных. Нам нужен драйвер SQL Server (как правило, он расположен в самом конце списка). После этого нажмите **NEXT.**

Во втором окне необходимо ввести имя файла, в котором будет сохранена информация об источнике данных. (**NEXT**)

В третьем окне необходимо нажать кнопку **FINISH**. После этого создается указанный выше файл и далее идет его настройка. (**NEXT**)

В четвертом окне необходимо заполнить только строку **Сервер** – из ниспадающего списка выбрать имя сервера сети. (На нашей кафедре один сервер, поэтому в списке будет всегда одно имя) (**NEXT**)

В пятом окне все настройки остаются без изменений. Здесь предлагается выбрать способ проверки подлинности пользователя. Этот вопрос я не рассматривал, всегда использовал настройки по- умолчанию (**NEXT**)

В шестом окне необходимо задать имя базы данных на SQL Server, которая будет использоваться в качестве источника данных. Для этого нужно установить флажок «Использовать по умолчанию базу данных» и из списка выбрать нужную базу данных. В списке каждый раз будут находиться несколько названий системных баз данных SQL, которые выбирать не надо, т.к. они используются для работы самого SQL Server. Итак, в данном окне производится выбор базы данных, данные с которой необходимы пользователю.(**NEXT**)

В седьмом окне настройки также остаются без изменений. После нажатия кнопки **FINISH** выводится окно с настройками созданного источника данных. После нажатия на кнопку «Проверка источника данных» происходит пробное соединение с сервером и если все сделано правильно выводится надпись: TESTS COMPLETED SUCCESSFULLY!.

После этого в окне администратора источников данных ODBC на закладке «Файловый DSN» в списке появиться имя созданного файла, содержащего описание источника данных. На этом создание источника данных завершено, окно администратора источников данных необходимо закрыть.

Создание запроса к серверу в среде Access

Запустить Access

В открывшемся окне выбрать пункт «Новая база данных» и нажать **ОК**

В следующем окне необходимо ввести название файла, в котором будет храниться создаваемая база данных и нажать кнопку «Создать». После этого отроется окно базы данных.

В окне базы данных выбрать пункт «Запросы» в левой части окна (для Access 2000), затем нажать кнопку **«Создать»**

В появившемся списке выбрать пункт **«Конструктор»** и нажать ОК. Автоматически запускается окно конструктора запросов. Помимо этого запускается окно «Добавление таблицы», которое необходимо закрыть.

В меню **«Запрос»** выбрать пункт **«Запрос SQL»** и в выпадающем списке выбрать пункт **«Запрос к серверу»**. Автоматически выводится окно редактора запросов SQL, в котором необходимо написать запрос.

После этого необходимо закрыть окно редактора запросов. При этом появится предупреждение с предложением сохранить данный запрос (нажать **«Да»**), далее необходимо будет ввести имя запроса.

Если все эти операции выполнены, то в окне базы данных появится имя созданного запроса с пиктограммой в виде глобуса. Такой пиктограммой обозначаются запросы к серверу. На этом создание запроса завершено.

Запуск запроса из среды Access

Для запуска запроса необходимо два раза нажать на его имя в окне базы данных мышкой. После этого запускается окно выбора источника данных. В этом окне необходимо выбрать закладку «Файловый источник данных». В центральной части будут расположены имена заранее созданных источников данных (см. раздел «Установка связи между Access и SQL Server.»). Для продолжения работы запроса необходимо выбрать источник данных и нажать ОК. Автоматически запустится окно «Вход в сервер SQL».

Удобнее использовать доверительное соединение (**Trusted connection**), для чего необходимо установить в окне «Вход в сервер SQL» соответствующий флажок и нажать ОК. После этого происходит связь с сервером и запрос выполняется.