В. М. Илюшечкин

ОСНОВЫ ИСПОЛЬЗОВАНИЯ И ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ

УЧЕБНИК ДЛЯ СПО

Рекомендовано Учебно–методическим отделом среднего профессионального образования в качестве учебника для студентов образовательных учреждений среднего профессионального образования

Книга доступна в электронной библиотечной системе biblio-online.ru

Москва ▪ Юрайт ▪ 2019

УДК 004.65(075.32)
ББК 32.973-018.2я723
И43

***Автор*:**

Илюшечкин Владимир Михайлович — кандидат технических наук, доцент кафедры информатики и программного обеспечения вычислительных систем факультета микроприборов и технической кибернетики Национального исследовательского университета «МИЭТ». Автор ряда научных трудов по программированию.

Илюшечкин, В. М.

И43 Основы использования и проектирования баз данных : учебник для СПО / В. М. Илюшечкин. — М. : Издательство Юрайт, 2019. — 213 с. — Серия : Профессиональное образование.

ISBN 978-5-534-01283-5

В учебнике содержатся теоретические и практические сведения о современных системах управления базами данных (СУБД), об использовании и проектировании баз данных. Рассматриваются языковые и программные средства СУБД и систем автоматизации проектирования баз данных. Приведены примеры создания инфологических и даталогических моделей, позволяющие студентам научиться проектировать базы данных.

Соответствует актуальным требованиям Федерального государственного образовательного стандарта среднего профессионального образования и профессиональным требованиям.

Для студентов образовательных учреждений среднего профессионального образования, обучающихся по специальности «Информатика и вычислительная техника».

УДК 004.65(075.32)
ББК 32.973-018.2я723

ISBN 978-5-534-01283-5

Оглавление

Принятые сокращения

БД — база данных
БнД — банк данных
ДЛМ — даталогическая модель базы данных
ИЛМ — инфологическая модель предметной области
ИМ — иерархическая модель данных
НФ — нормальная форма отношения
НФБК — нормальная форма Бойса — Кодда
ОЦ — ограничения целостности
ПО — предметная область
РБД — реляционная база данных
РМ — реляционная модель данных
СМ — сетевая модель данных
СП — сущность-потомок
СР — сущность-родитель
СУБД — система управления базами данных
ФЗ — функциональная зависимость
ФМ — физическая модель базы данных
ЯОД — язык описания данных
3GL — Third-Generation Language (язык третьего поколения)
4GL — Fourth-Generation Language (язык четвертого поколения)
ANSI — American National Standards Institute (Американский институт национальных стандартов)
API — Application Programming Interface (интерфейс прикладного программирования)
ASP — Active Server Pages
BDE — машина баз данных Borland Database Engine
CASE — Computer Aided Software Engineering
CGI — Common Gateway Interface
DDL — Data Definition Language (язык определения данных)
DFD — Data Flow Diagram
DML — Data Manipulation Language (язык манипулирования данными)

DQL — Data Query Language (язык запросов данных)
ER — Entity-Relationship (сущность-связь)
GUID — Globally Unique Identifier (уникальный идентификационный номер)
HTML — Hypertext Markup Language (язык разметки гипертекста)
HTTP — Hypertext Transfer Protocol (сетевой протокол передачи гипертекста)
ICAM — Integrated Computer Aided Manufacturing (интегрированная компьютеризация производства)
IDEF — методология ICAM DEFinition
IE — методология Information Engineering
IIS — Web-сервер Microsoft Internet Information Services
ISO — International Organization for Standardization (Международная организация по стандартам)
MSDE — машина баз данных Microsoft Data Engine
OLE — Object Linking and Embedding (технология связывания и внедрения объектов и протокол разработанные компанией «Майкрософт»)
OLTP — Online Transaction Processing (оперативная обработка транзакций)
PHP — Hypertext Preprocessor (Препроцессор Гипертекста — скриптовый язык программирования)
QBE — Query-By-Example (язык запросов по образцу)
SADT — Structured Analysis and Design Technique (метод структурного анализа и проектирования)
SGML — Standard Generalized Markup Language (стандартный общий язык разметки)
SPARC — Standards Planning and Requirements Committee (подкомитет Американского института национальных стандартов)
SQL — Structured Query Language (структурированный язык запросов)
UML — Unified Modeling Language (унифицированный язык моделирования)
URL — Uniform Resource Locator (определитель местонахождения информационного ресурса)
WWW — World Wide Web (Всемирная паутина)
XML — eXtensible Markup Language (расширяемый язык разметки)

Предисловие

Потребность в информации стала одной из самых насущных в жизни современного цивилизованного человечества. Включая утром радиоприемники и телевизоры, люди с нетерпением ждут новостей о погоде, курсах валют, сообщений о показателях деловой активности и т.д. Приходя на работу, они погружаются в море деловой информации, которую получают через свои служебные компьютеры. Возвращаясь домой, они заходят в супермаркеты за покупками и из кассовых чеков узнают информацию о цене приобретенных товаров.

Источником значительной части информации являются базы данных, в которых содержатся сведения о прогнозах погоды, ежедневных курсах валют, показателях произведенной продукции, ценах и количестве продаваемых товаров, расписаниях движения поездов, отправлении и прибытии самолетов и т.д. Структура баз данных может мало интересовать конечного пользователя, поскольку для него более важны время получения информации, ее актуальность и доступный объем. Создание базы данных, отвечающей всем требованиям пользователей, становится задачей проектировщика, который должен обладать теоретическими знаниями и практическими навыками в области информационных технологий.

Согласно образовательным стандартам среднего профессионального образования подготовка специалиста по направлению «Информатика и вычислительная техника» предусматривает изучение дисциплины «Базы данных», по другим техническим направлениям раздел, посвященный базам данных, включен в общий курс информатики.

Предметом дисциплины «Базы данных» являются база данных как форма организации информационного ядра любой информационной системы, а также языковые и программные средства для работы с базами данных и методы проектирования баз данных. В учебном пособии содержатся основные сведения по этим темам, изложенные в шести главах.

В гл. 1 рассматриваются различные подходы к хранению данных, классификация и архитектура баз данных и систем управления базами данных, а также дается общее описание процесса проектирования баз данных.

Глава 2 посвящена реляционной модели данных, которая служит основой большинства современных баз данных.

В гл. 3 представлены языки, предназначенные для работы с базами данных. Подробно описывается язык SQL, являющийся общепринятым языком взаимодействия с базами данных. Менее детально изло-

жены возможности языка QBE, относящегося к табличных языкам запросов.

В гл. 4 содержатся сведения о таких широко используемых системах управления базами данных, как Access, Oracle, SQL Server и InterBase, с указанием их технических характеристик и поддерживаемых типов данных.

Главы 5 и 6 посвящены вопросам проектирования баз данных. В гл. 5 излагается метод, основанный на принципах нормализации, и приводится пример использования этого метода для получения базы данных, соответствующей нормальной форме Бойса — Кодда.

В гл. 6 представлен инженерный подход к разработке базы данных, включающий этапы концептуального и даталогического проектирования, поддерживаемые существующими системами автоматизации проектирования баз данных, ряду которых дается краткая характеристика. В качестве примера в этой главе описаны приемы работы с одной из таких систем — ERwin.

Подходы, применяемые при использовании и построении баз данных, универсальны и мало зависят от предметной области, информация о которой хранится в базах данных. Чтобы рассматриваемые в пособии примеры были понятны студентам, выбраны предметные области, связанные с повседневной деятельностью людей.

В результате изучения материалов курса «Основы использования и проектирования баз данных» студент должен освоить:

трудовые действия

- навыки проектирования БД с использованием современных CASE-средств;
- навыки разработки и администрирования БД в среде современной СУБД;

необходимые умения

- проектировать БД реляционного типа;
- проводить нормализацию БД;
- писать запросы на языках SQL, DDL, DML;
- средствами современных СУБД провести преобразования реляционных данных в формат XML и обратно, писать запросы к XML-данным;

необходимые знания

- базовые понятия теории баз данных;
- основные модели данных;
- нормальные формы реляционных отношений;
- язык структурированных запросов SQL;
- особенности создания и использования программируемых объектов баз данных;
- способы совместного использования реляционных данных и данных в формате XML.

Глава 1

ОСНОВНЫЕ СВЕДЕНИЯ О ХРАНЕНИИ ДАННЫХ

1.1. Файловые системы хранения данных

Использование компьютерной техники связано с двумя большими областями деятельности человечества [10]. В первой области компьютерная техника применяется для выполнения численных расчетов, которые слишком долго или вообще невозможно производить вручную. Характерной особенностью данной области является наличие сложных алгоритмов обработки, которые применяются к простым по структуре данным, объем которых сравнительно невелик.

Вторая область включает в себя автоматические или автоматизированные информационные системы, представляющие собой программно-аппаратный комплексы, обеспечивающие надежное хранение информации в памяти компьютера, выполнение специфических для данного приложения преобразований информации и вычислений, предоставление пользователям удобного и легко осваиваемого интерфейса.

Обычно такие системы имеют дело с большими объемами информации, имеющей достаточно сложную структуру. Классическими примерами информационных систем являются банковские системы, автоматизированные системы управления предприятиями, системы резервирования авиационных или железнодорожных билетов и т.д.

Вторая область использования вычислительной техники возникла несколько позже первой. Это связано с тем, что в начальный период развития компьютеров их возможности по хранению информации были ограниченными. Поэтому важным шагом в развитии информационных систем явился переход к использованию централизованных систем управле-

ния файлами. С точки зрения прикладной программы файл — это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные. Правила именования файлов, способ доступа к данным, хранящимся в файле, и структура этих данных зависят от конкретной системы управления файлами и от типа файла. Система управления файлами обеспечивает распределение внешней памяти, отображение имен файлов в соответствующие адреса во внешней памяти и обеспечение доступа к данным.

Для пользователя файл представляется как линейная последовательность записей, с которой можно выполнять ряд стандартных операций:

— создать файл (требуемого типа и размера);
— открыть ранее созданный файл;
— прочитать из файла некоторую запись (текущую, следующую, предыдущую, первую, последнюю);
— записать в файл на место текущей записи новую;
— добавить новую запись в конец файла.

Структура записи файла была известна только программе, которая с ним работала, но не системе управления файлами, и поэтому для того, чтобы извлечь некоторую информацию из файла, необходимо было точно знать структуру записи файла. Каждая программа, работающая с файлом, должна была иметь у себя внутри структуру данных, соответствующую структуре этого файла. При изменении структуры файла требовалось изменять структуру программы, а это требовало новой компиляции, то есть процесса перевода программы в исполняемые машинные команды. Такая ситуации характеризовалась как зависимость программ от данных. Для информационных систем характерным является наличие большого числа различных пользователей (программ), каждый из которых имеет свои специфические алгоритмы обработки информации, хранящейся в одних и тех же файлах. Изменение структуры файла, которое было необходимо для одной программы, требовало исправления и перекомпиляции и дополнительной отладки всех остальных программ, работающих с этим же файлом. Это было первым существенным недостатком файловых систем, который явился толчком к созданию новых систем хранения и управления информацией.

Поскольку файловые системы являются общим хранилищем файлов, принадлежащих разным пользователям, системы управления файлами должны обеспечивать авторизацию доступа к файлам, чтобы по отношению к каждому заре-

гистрированному пользователю данной информационной системы для каждого существующего файла указывались действия, которые разрешены или запрещены конкретному пользователю. Администрирование режимом доступа к файлу в основном выполнялось его создателем — владельцем. Для множества файлов, отражающих информационную модель одной предметной области, такой децентрализованный принцип управления доступом вызывал дополнительные трудности. Отсутствие централизованных методов управления доступом к информации послужило одной из причин разработки систем управления базами данных (СУБД).

Следующей причиной, способствовавшей созданию СУБД, стала необходимость обеспечения эффективной параллельной работы многих пользователей с одними и теми же файлами. В общем случае системы управления файлами обеспечивали режим многопользовательского доступа. Если операционная система поддерживает многопользовательский режим, вполне реальна ситуация, когда не менее двух пользователей одновременно пытаются работать с одним и тем же файлом. Если все пользователи собираются только читать файл, ничего страшного не произойдет. Но если хотя бы один из них будет изменять файл, для корректной работы этих пользователей требуется взаимная синхронизация их действий по отношению к файлу.

Ограничениями, присущими файловым системам, являются разделение и изоляция данных, дублирование данных, зависимость от данных, несовместимость файлов, фиксированные запросы и быстрое увеличение количества приложений [13].

Когда данные изолированы в отдельных файлах, доступ к ним весьма затруднителен. Для извлечения соответствующей поставленным условиям информации программист должен организовать синхронную обработку двух или более файлов.

Из-за децентрализованной работы с данными, проводимой пользователями независимо друг от друга, в файловой системе фактически допускается бесконтрольное дублирование данных, которое нежелательно по следующим причинам:

— дублирование данных сопровождается неэкономным расходованием ресурсов, поскольку на ввод избыточных данных требуется затрачивать дополнительное время и деньги;

— для их хранения необходимо дополнительное место во внешней памяти, что связано с дополнительными накладными расходами;

— дублирование данных может привести к нарушению их целостности.

Физическая структура и способ хранения записей файлов данных жестко зафиксированы в коде приложений. Это значит, что изменить существующую структуру данных достаточно сложно. Кроме того, при модификации структуры данных программы должны быть изменены с целью соответствия новой структуре файла. А таких программ может быть очень много. Следовательно, программист должен прежде всего выявить все программы, нуждающиеся в доработке, а затем их перепроверить и изменить. Ясно, что выполнение всех этих действий требует больших затрат времени и может явиться причиной появления ошибок. Данная особенность файловых систем называется зависимостью программ от данных.

Поскольку структура файлов определяется кодом приложений, она также зависит от языка программирования этого приложения. Например, структура файла, созданного программой на языке Паскаль, может значительно отличаться от структуры файла, создаваемого программой на языке С. Прямая несовместимость таких файлов затрудняет процесс их совместной обработки и может потребовать создания программного обеспечения, предназначенного для преобразования полей файлов в некоторый общий формат, поэтому этот процесс может оказаться весьма длительным и дорогим.

С точки зрения пользователя возможности файловых систем намного превосходят возможности ручных операций с информацией. Соответственно возрастают и требования к реализации новых или модифицированных запросов. Однако файловые системы требуют больших затрат труда программиста, поскольку все необходимые запросы и отчеты должны быть созданы именно им. Во-первых, во многих организациях типы применяемых запросов и отчетов имели фиксированную форму, и не было никаких инструментов создания незапланированных или произвольных запросов как к самой информации, так и к сведениям о том, какие типы информации доступны.

Во-вторых, в других организациях наблюдалось быстрое увеличение количества файлов и приложений. В конечном счете наступал момент, когда программное обеспечение было неспособно адекватно отвечать запросам пользователей,

эффективность его снижалась, а недостаточность документирования имела следствием дополнительное усложнение сопровождения программ. При этом часто игнорировались вопросы поддержки функционирования системы: не предусматривались меры по обеспечению безопасности или целостности данных; средства восстановления в случае сбоя аппаратного или программного обеспечения были крайне ограничены или вообще отсутствовали. Доступ к файлам часто ограничивался узким кругом пользователей, т.е. не предусматривалось их совместное использование даже сотрудниками одного и того же отдела.

1.2. Системы с использованием баз данных

Все перечисленные выше ограничения файловых систем связаны с определением данных внутри приложений вместо независимого хранения от них и отсутствием других инструментов доступа к данным и их обработки помимо приложений.

Существовавшие ограничения заставили разработчиков информационных систем предложить новый подход к управлению информацией. Этот подход был реализован в рамках программных систем, названных впоследствии системами управления базами данных, а сами хранилища информации, которые работали под управлением СУБД, назывались базами или банками данных.

База данных (БД) — это поименованная совокупность взаимосвязанных данных, управляемых специальной системой, называемой СУБД.

СУБД представляет собой совокупность специальных языковых и программных средств, облегчающих пользователям выполнение всех операций, связанных с организацией хранения данных, их корректировкой и доступом к ним. СУБД служит, по существу, посредником между пользователем и БД.

БД и СУБД являются составными частями более сложной системы, именуемой банком данных [6]. *Банк данных (БнД)* — это система, состоящая из баз данных, программных, технических, языковых, организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования

данных. Термин «банк данных» схож с понятием «система баз данных». *Система баз данных* представляет собой совокупность программного обеспечения, данных и аппаратного обеспечения компьютеров, которая реализует набор приложений и моделей данных, и использует СУБД и прикладное программное обеспечение для создания конкретной информационной системы [18]

БнД является сложной человеко-машинной системой,

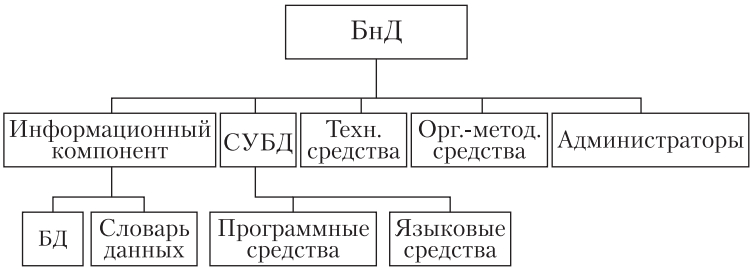


Рис. 1.1. **Компоненты БнД**

состоящей из взаимосвязанных и взаимозависимых компонентов (рис. 1.1).

Информационный компонент БнД содержит базу данных и словарь данных. Словарь данных является хранилищем метаинформации, т.е. информации об информации. Метаинформация включает в себя описание базы данных, информацию о предметной области, представленной в БД, сведения о пользователях БнД, а также некоторую другую информацию.

СУБД предоставляет пользователю программные и языковые средства, обеспечивающие взаимодействие всех частей информационной системы при ее функционировании.

Языковые средства СУБД относятся к языкам четвертого поколения. Языковые средства предназначаются для пользователей разных категорий: конечных пользователей, системных аналитиков, профессиональных программистов. По функциональным возможностям выделяются 10 категорий языков [6]. По форме представления различают аналитические, табличные и графические языковые средства.

Некоторые СУБД предоставляют пользователю несколько языков для достижения одной и той же цели. Например, в системе Delphi [19] для работы с базами данных могут использоваться:

— процедурный язык Object Pascal;
— табличный язык запросов QBE (Query-By-Example);
— язык SQL (Structured Query Language).

В системе FoxPro доступны процедурный язык xBase, табличный язык запросов Relation QBE, а также язык SQL.

Технические средства, используемые в БнД, — это компьютеры различных классов (от больших до персональных), периферийные устройства, коммуникационная (сетевая) аппаратура.

Организационно-методические средства представляют собой различные инструкции, методические и регламентирующие материалы, предназначенные для пользователей, взаимодействующих с БнД.

Администраторы — это специалисты, которые обеспечивают создание, функционирование и развитие БнД. База данных и СУБД являются коллективными ресурсами, которыми следует управлять так же, как и любыми другими ресурсами [13]. Обычно управление данными и базой данных предусматривает управление и контроль за СУБД и помещенными в нее данными. *Администратор данных* отвечает за управление данными, включая планирование базы данных, разработку и сопровождение стандартов, прикладных алгоритмов и деловых процедур, а также за концептуальное и логическое проектирование базы данных. *Администратор базы данных* отвечает за физическую реализацию базы данных, включая физическое проектирование и воплощение проекта, за обеспечение безопасности и целостности данных, за сопровождение операционной системы, а также за обеспечение максимальной производительности приложений и пользователей. По сравнению с администратором данных обязанности администратора базы данных носят более технический характер, и для него необходимо знание конкретной СУБД и системного окружения. В одних организациях между этими ролями не делается различий, а в других важность коллективных ресурсов отражена именно в выделении отдельных групп персонала с указанным кругом обязанностей.

В проектировании больших баз данных участвуют разработчики двух разных типов: *разработчики логической структуры базы данных* и *разработчики физической структуры базы данных*. Разработчик логической структуры базы данных занимается идентификацией данных (т.е. сущностей и их атрибутов), связей между данными и устанавливает ограничения, накладываемые на хранимые данные. Разработчик

логической структуры базы данных должен обладать всесторонним и полным пониманием структуры данных организации и процессов ее функционирования.

Для эффективной работы разработчик логической структуры базы данных должен как можно раньше вовлечь всех предполагаемых пользователей базы данных в процесс создания модели данных. Деятельность разработчика логической структуры базы данных делится на две части

— концептуальное проектирование базы данных, которое совершенно не зависит от таких деталей ее воплощения, как конкретная целевая СУБД, приложения, языки программирования или любые другие физические характеристики;

— даталогическое проектирование базы данных, которое проводится с учетом особенностей выбранной модели данных.

Разработчик физической структуры базы данных получает готовую логическую модель данных и занимается ее физической реализацией, в том числе:

— преобразованием логической модели данных в набор таблиц и ограничений целостности данных;

— выбором конкретных структур хранения и методов доступа к данным, обеспечивающих необходимый уровень производительности при работе с базой данных;

—проектированием любых требуемых мер защиты данных.

Многие этапы физического проектирования базы данных в значительной степени зависят от выбранной целевой СУБД, а потому может существовать несколько различных способов воплощения требуемой схемы. Следовательно, разработчик физической структуры базы данных должен разбираться в функциональных возможностях целевой СУБД и понимать достоинства и недостатки каждого возможного варианта реализации. Разработчик физической структуры базы данных должен уметь выбрать наиболее подходящую стратегию хранения данных с учетом всех существующих особенностей их использования.

После создания базы данных начинается разработка приложений, предоставляющих пользователям необходимые им функциональные возможности. Именно эту работу и выполняют *прикладные программисты*. Обычно прикладные программисты работают на основе спецификаций, созданных системными аналитиками. Как правило, каждая программа содержит некоторые операторы, требующие от СУБД выполнения определенных действий с базой данных — например таких, как извлечение, вставка, обновление или удаление дан-

ных. Эти программы могут создаваться на различных языках программирования третьего или четвертого поколения.

Конечные пользователи являются клиентами базы данных; она проектируется, создается и поддерживается для того, чтобы обслуживать их информационные потребности. Пользователей можно классифицировать по способу использования ими системы на рядовых и опытных.

Рядовые пользователи обычно и не подозревают о наличии СУБД. Они обращаются к базе данных с помощью специальных приложений, позволяющих в максимальной степени упростить выполняемые ими операции. Такие пользователи инициируют выполнение операций базы данных, вводя простейшие команды или выбирая команды меню. Это значит, что таким пользователям не нужно ничего знать о базе данных или СУБД. Например, чтобы узнать цену товара, кассир в супермаркете использует сканер для считывания нанесенного на него штрих-кода. В результате этого простейшего действия специальная программа не только считывает штрих-код, но и выбирает на основе его значения цену товара из базы данных, а также уменьшает значение в другом поле базы данных, обозначающем остаток таких товаров на складе, после чего выбивает цену и общую стоимость на кассовом аппарате.

Опытные пользователи знакомы со структурой базы данных и возможностями СУБД. Для выполнения требуемых операций они могут использовать такой язык запросов высокого уровня, как SQL. А некоторые опытные пользователи могут даже создавать собственные прикладные программы.

Использование СУБД для доступа к данным дает ряд преимуществ, к которым относятся [13]:

— контроль за избыточностью данных;
— непротиворечивость данных;
— больше полезной информации при том же объеме хранимых данных;
— совместное использование данных;
— поддержка целостности данных;
— повышенная безопасность;
— применение стандартов;
— повышение эффективности с ростом масштабов системы;
— возможность нахождения компромисса при противоречивых требованиях;
— повышение доступности данных и их готовности к работе;

— улучшение показателей производительности;
— упрощение сопровождения системы за счет независимости от данных;
— улучшенное управление параллельной работой;
— развитые службы резервного копирования и восстановления.

Контроль за избыточностью данных. Как отмечалось выше, традиционные файловые системы неэкономно расходуют внешнюю память, сохраняя одни и те же данные в нескольких файлах. При использовании базы данных, наоборот, предпринимается попытка исключить избыточность данных за счет интеграции файлов, что позволяет отказаться от хранения нескольких копий одного и того же элемента информации. Однако полностью избыточность информации в базах данных не исключается, а лишь ограничивается ее степень. В одних случаях ключевые элементы данных необходимо дублировать для моделирования связей, а в других случаях некоторые данные требуется дублировать из соображений повышения производительности системы.

Непротиворечивость данных. Устранение избыточности данных или контроль над ней позволяет уменьшить риск возникновения противоречивых состояний. Если элемент данных хранится в базе только в одном экземпляре, то для изменения его значения потребуется выполнить только одну операцию обновления, причем новое значение станет доступным сразу всем пользователям базы данных. А если этот элемент данных с ведома системы хранится в базе данных в нескольких экземплярах, то такая система сможет следить за тем, чтобы копии не противоречили друг другу. К сожалению, во многих современных СУБД такой способ обеспечения непротиворечивости данных не поддерживается автоматически.

Больше полезной информации при том же объеме хранимых данных. Благодаря объединению рабочих данных предприятия на основе тех же данных можно получать дополнительную информацию. Например, в файловой системе сотрудникам отдела кадров недоступны сведения об окладах работников предприятия. Аналогично, сотрудники расчетного отдела бухгалтерии не имеют полных сведений семейном положении работников. При объединении этих файлов в общей базе сотрудники обоих отделов смогут получать больше информации.

Совместное использование данных. Файлы обычно принадлежат отдельным лицам или целым отделам, которые

используют их в своей работе. В то же время база данных принадлежит всему предприятию в целом и может совместно использоваться всеми зарегистрированными пользователями. При такой организации работы большее количество пользователей может работать с большим объемом данных. Более того, при этом можно создавать новые приложения на основе уже существующей в базе данных информации и добавлять в нее только те данные, которые в настоящий момент еще не хранятся в ней, а не определять заново требования ко всем данным, необходимым новому приложению. Новые приложения могут также использовать такие предоставляемые типичными СУБД функциональные возможности, как определение структур данных и управление доступом к данным, организация параллельной обработки и обеспечение средств копирования/восстановления, исключив необходимость реализации этих функций со своей стороны.

Поддержка целостности данных. Целостность базы данных означает корректность и непротиворечивость хранимых в ней данных. Целостность обычно описывается с помощью ограничений, т.е. правил поддержки непротиворечивости, которые не должны нарушаться в базе данных. Ограничения можно применять к элементам данных внутри одной записи или к связям между записями. Например, ограничение целостности может гласить, что оклад сотрудника не должен превышать 20 тыс. руб. в месяц или же что в записи с данными о сотруднике номер подразделения, в котором он работает, должен соответствовать реально существующему в компании. Таким образом, интеграция данных позволяет администратору баз данных задавать требования по поддержке целостности данных, а системе управления базами данных применять их.

Повышенная безопасность. Безопасность базы данных заключается в защите базы данных от несанкционированного доступа со стороны пользователей. Без привлечения соответствующих мер безопасности объединенные данные становятся более уязвимыми, чем данные в файловой системе. Однако объединение позволяет администратору баз данных определить требуемую систему безопасности базы данных, а СУБД привести ее в действие. Система обеспечения безопасности может быть выражена в форме имен и паролей для идентификации пользователей, которые зарегистрированы в этой базе данных. Доступ к данным со стороны зарегистрированного пользователя может быть ограничен только

некоторыми операциями (извлечением, вставкой, обновлением и удалением).

Применение стандартов. Объединение данных позволяет администратору баз данных определять и применять необходимые стандарты. Например, стандарты предприятия, государственные и международные стандарты могут регламентировать формат данных при обмене ими между системами, соглашения об именах, форму представления документации, процедуры обновления и правила доступа.

Повышение эффективности с увеличением масштабов системы. Комбинируя все рабочие данные предприятия в одной базе данных и создавая приложения, которые работают с одним источником данных, можно добиться существенной экономии средств. В этом случае бюджет, который обычно выделялся каждому отделу для разработки и поддержки их собственных файловых систем, можно объединить с бюджетами других подразделений (с более низкой общей стоимостью), что позволит добиться повышения эффективности при росте масштабов производства. Теперь консолидированный бюджет можно будет использовать для приобретения оборудования в той конфигурации, которая в большей степени отвечает потребностям предприятия.

Возможность нахождения компромисса для противоречивых требований. Потребности одних пользователей или подразделений предприятия могут противоречить потребностям других пользователей. Но поскольку база данных контролируется администратором баз данных, он может принимать решения о проектировании и способе использования базы данных, при которых имеющиеся ресурсы всего предприятия в целом будут использоваться наилучшим образом. Эти решения обеспечивают оптимальную производительность для самых важных приложений, причем чаще всего за счет менее критичных.

Повышение доступности данных и их готовности к работе. Данные, необходимость в которых не ограничивается рамками отдельных подразделений, в результате объединения становятся непосредственно доступными конечным пользователям. Потенциально это повышает функциональность системы, что, например, может быть использовано для более качественного обслуживания конечных пользователей или клиентов предприятия. Во многих СУБД предусмотрены языки запросов или инструменты для создания отчетов, которые позволяют пользователям вводить не предусмотренные

заранее запросы и почти немедленно получать требуемую информацию на своих терминалах, не прибегая к помощи программиста, который для извлечения этой информации из базы данных должен был бы создать специальное программное обеспечение.

Улучшение показателей производительности. Поскольку в СУБД предусмотрено много стандартных функций, которые программист обычно должен самостоятельно реализовать в приложениях для файловых систем, то на базовом уровне СУБД обеспечивает все низкоуровневые процедуры работы с файлами, которую обычно выполняют приложения. Наличие этих процедур позволяет программисту сконцентрироваться на разработке более специальных, необходимых пользователям функций, не заботясь о подробностях их воплощения на более низком уровне. Во многих СУБД предусмотрена также среда разработки с инструментами, упрощающими создание приложений баз данных. Результатом является повышение производительности работы программистов и сокращение времени разработки новых приложений.

Упрощение сопровождения системы за счет независимости от данных. В файловых системах описания данных и логика доступа к данным встроены в каждое приложение, поэтому программы становятся зависимыми от данных. Для изменения структуры данных или для изменения способа хранения данных на диске может потребоваться существенно преобразовать все программы, на которые эти изменения способны оказать влияние. В СУБД подход иной: описания данных отделены от приложений, а потому приложения защищены от изменений в описаниях данных. Эта особенность называется независимостью от данных. Наличие независимости программ от данных значительно упрощает обслуживание и сопровождение приложений, работающих с базой данных.

Улучшенное управление параллельной работой. В некоторых файловых системах при одновременном доступе к одному и тому же файлу двух пользователей может возникнуть конфликт двух запросов, результатом которого будет потеря информации или утрата ее целостности. В свою очередь, во многих СУБД предусмотрена возможность параллельного доступа к базе данных и гарантируется отсутствие подобных проблем.

Развитые службы резервного копирования и восстановления. Ответственность за обеспечение защиты данных от

сбоев аппаратного и программного обеспечения в файловых системах возлагается на пользователя. Так, может потребоваться каждую ночь выполнять резервное копирование данных. При этом в случае сбоя может быть восстановлена резервная копия, но результаты работы, выполненной после резервного копирования, будут утрачены, и данную работу потребуется выполнить заново. В современных СУБД предусмотрены средства снижения вероятности потерь информации при возникновении различных сбоев.

Однако использование СУБД дает не только преимущества, но сопряжено и с такими негативными последствиями, как:

— сложность СУБД;
— большой размер СУБД;
— значительная стоимость некоторых СУБД;
— дополнительные затраты на аппаратное обеспечение;
— затраты на преобразование имеющейся инфраструктуры предприятия;
— снижение производительности;
— более серьезные последствия при выходе системы из строя.

Сложность. Обеспечение функциональности, которой должна обладать каждая хорошая СУБД, сопровождается значительным усложнением программного обеспечения СУБД. Чтобы воспользоваться всеми преимуществами СУБД, проектировщики и разработчики баз данных, администраторы данных и администраторы баз данных, а также конечные пользователи должны хорошо понимать функциональные возможности СУБД. Непонимание принципов работы системы может привести к неудачным результатам проектирования, что будет иметь самые серьезные последствия для всего предприятия.

Размер. Сложность и широта функциональных возможностей приводят к тому, что СУБД становится чрезвычайно сложным программным продуктом, который может потребовать много места на диске и нуждаться в большом объеме оперативной памяти для эффективной работы.

Стоимость СУБД. В зависимости от имеющейся вычислительной среды и требуемых функциональных возможностей стоимость СУБД может изменяться в очень широких пределах. Например, однопользовательская СУБД для персонального компьютера может стоить около 100 долл. Однако большая многопользовательская СУБД для мощного

серверного компьютера, обслуживающая сотни пользователей, может быть чрезвычайно дорогостоящей: от 100 000 до 1 000 000 долл. Кроме того, следует учесть ежегодные расходы на сопровождение системы, которые составляют некоторый процент от ее общей стоимости.

Дополнительные затраты на аппаратное обеспечение. Для удовлетворения требований, предъявляемых к дисковым накопителям со стороны СУБД и базы данных, может понадобиться приобрести дополнительные устройства хранения информации. Более того, для достижения требуемой производительности может понадобиться более мощный компьютер, который, возможно, будет работать только с СУБД. Приобретение другого дополнительного аппаратного обеспечения приведет к дальнейшему росту затрат.

Затраты на преобразование. В некоторых ситуациях стоимость СУБД и дополнительного аппаратного обеспечения может оказаться несущественной по сравнению со стоимостью преобразования существующих приложений для работы с новой СУБД и новым аппаратным обеспечением. Эти затраты включают также стоимость подготовки персонала для работы с новой системой, а также оплату услуг специалистов, которые будут оказывать помощь в преобразовании и запуске новой системы. Все это является одной из основных причин, по которой некоторые предприятия остаются сторонниками прежних систем и не хотят переходить к более современным технологиям управления базами данных. Термин «традиционная система» иногда используется для обозначения устаревших и, как правило, не самых лучших систем.

Производительность. Обычно файловая система создается для некоторых специализированных приложений, например для оформления заказов, а потому ее производительность может быть весьма высока. Однако СУБД предназначены для решения более общих задач и обслуживания сразу нескольких приложений, а не какого-то одного из них. В результате многие приложения в новой среде будут работать не так быстро, как прежде.

Более серьезные последствия при выходе системы из строя. Централизация ресурсов повышает уязвимость системы. Поскольку работа всех пользователей и приложений зависит от готовности к работе СУБД, выход из строя одного из ее компонентов может привести к полному прекращению всей работы предприятия.

1.3. Архитектура баз данных

Построению систем, основанных на использовании СУБД, предшествовали интенсивные научные исследования, направленные на решение проблемы устройства самой СУБД. Наиболее фундаментальным результатом этих исследований стало определение трех уровней абстракции, т.е. трех различных уровней описания элементов данных, зафиксированных в модели ANSI-SPARC [13]. Эти уровни формируют трехуровневую архитектуру базы данных, которая охватывает внешний, концептуальный и внутренний уровни (рис. 1.2). Уровень, на котором данные воспринимаются пользователями, называется *внешним уровнем* (external level), тогда как СУБД и операционная система воспринимают данные на *внутреннем уровне* (internal level). Именно на внутреннем уровне данные реально сохраняются с использованием структур данных и файловой организации. *Концептуальный уровень* (conceptual level) представления данных предназначен для отображения внешнего уровня на внутренний и обеспечения необходимой независимости друг от друга.

Цель трехуровневой архитектуры заключается в отделении пользовательского представления базы данных от ее физического представления. Такое разделение базы данных на уровни объясняется следующими причинами:

— каждый пользователь должен иметь возможность обращаться к одним и тем же данным, реализуя свое собственное представление о них;

— каждый пользователь должен иметь возможность изменять свое представление о данных, причем это изменение не должно оказывать влияния на других пользователей;

— пользователи не должны непосредственно иметь дело с какими-либо подробностями физического хранения данных в базе, т.е. взаимодействие пользователя с базой не должно зависеть от особенностей хранения в ней данных;

— администратор базы данных должен иметь возможность изменять структуру хранения данных в базе, не оказывая влияния на пользовательские представления;

— внутренняя структура базы данных не должна зависеть от таких изменений физических аспектов хранения информации, как переключение на новое устройство хранения;

— администратор базы данных должен иметь возможность изменять концептуальную структуру базы данных без какого-либо влияния на всех пользователей.

Внешний уровень — это представление базы данных с точки зрения пользователей. Он описывает ту часть базы данных, которая относится к каждому пользователю. Внешний уровень состоит из нескольких различных внешних представлений базы данных. Каждый пользователь имеет дело с представлением «реального мира», выраженным в наиболее удобной для него форме. Внешнее представление содержит только те сущности, атрибуты и связи «реального мира», которые интересны пользователю. Другие сущности, атрибуты или связи, которые ему неинтересны, также могут быть представлены в базе данных, но пользователь может даже не подозревать об их существовании.

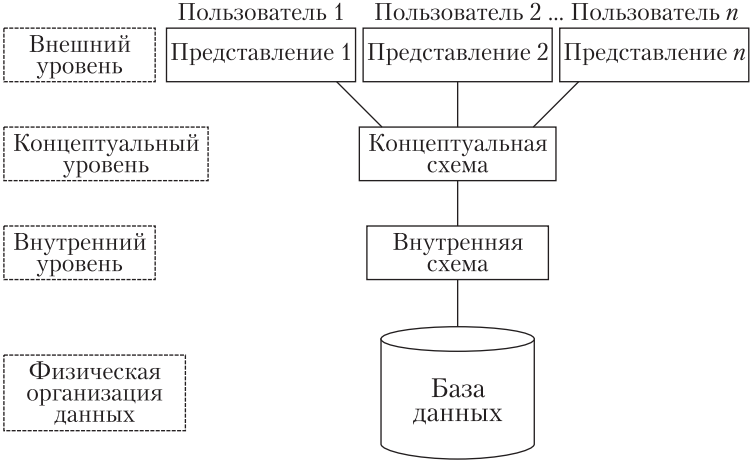


Рис. 1.2. **Трехуровневая архитектура ANSI-SPARC**

Помимо этого, различные представления могут по-разному отображать одни и те же данные. Например, один пользователь может просматривать даты в формате (день, месяц, год), а другой — в формате (год, месяц, день). Некоторые представления могут включать производные или вычисляемые данные, которые не хранятся в базе данных как таковые, а создаются по мере надобности. Представления могут также включать комбинированные или производные данные из нескольких объектов.

Концептуальный уровень соответствует обобщающему представлению базы данных. Этот уровень описывает то, какие данные хранятся в базе данных, а также связи, существующие между ними. Являясь промежуточным уровнем в трехуровневой архитектуре, он содержит логическую структуру всей базы данных (с точки зрения администратора базы данных). Фактически это полное представление требований к данным со стороны предприятия, которое не зависит от любых соображений относительно способа их хранения. На концептуальном уровне представлены следующие компоненты:

— все сущности, их атрибуты и связи;
— накладываемые на данные ограничения;
— информация о смысловом содержании данных;
— информация о мерах обеспечения безопасности и поддержки целостности данных.

Концептуальный уровень поддерживает каждое внешнее представление, в том смысле, что любые доступные пользователю данные должны содержаться (или могут быть вычислены) на этом уровне. Однако этот уровень не содержит никаких сведений о методах хранения данных. Например, описание сущности должно содержать сведения о типах данных атрибутов (целочисленный, действительный или символьный) и их длине (количестве значащих цифр или максимальном количестве символов), но не должно включать сведений об организации хранения данных, например об объеме занятого пространства в байтах.

Внутренний уровень отражает физическое представление базы данных в компьютере, описывая, как информация хранится в базе данных. Он описывает физическую реализацию базы данных и предназначен для достижения оптимальной производительности и обеспечения экономного использования дискового пространства. Он содержит описание структур данных и организации отдельных файлов, используемых для хранения данных на запоминающих устройствах. На этом уровне осуществляется взаимодействие СУБД с методами доступа операционной системы (вспомогательными функциями хранения и извлечения записей данных) с целью размещения данных на запоминающих устройствах, создания индексов, извлечения данных и т.д. На внутреннем уровне хранится следующая информация:

— распределение дискового пространства для хранения данных и индексов;

— описание подробностей сохранения записей (с указанием реальных размеров сохраняемых элементов данных);

— сведения о размещении записей;

— сведения о сжатии данных и выбранных методах их шифрования.

Ниже внутреннего уровня находится *физический уровень* (physical level), который реализуется операционной системой, но под управлением СУБД. Однако функции СУБД и операционной системы на физическом уровне не вполне четко разделены и могут отличаться от системы к системе. В одних СУБД используются многие предусмотренные в данной операционной системе методы доступа, тогда как в других применяются только самые основные и реализована собственная файловая организация. Физический уровень доступа к данным ниже СУБД состоит только из известных операционной системе элементов.

Трехуровневая архитектура баз данных отражается в последовательности этапов их проектирования. В базе данных отражается информация об определенной *предметной области (ПО)*, которой называется часть реального мира, представляющая интерес для данного исследования или использования.

Чтобы спроектировать структуру базы данных, необходима исходная информация о предметной области. Желательно, чтобы эта информация была представлена в формализованном виде.

Описание предметной области, выполненное без ориентации на используемые в дальнейшем СУБД и технические средства, называется инфологической моделью предметной области.

На основе ИЛМ строится даталогическая модель базы данных, которая представляет собой отображение логических связей между информационными элементами инфологической модели. Даталогическая модель строится в терминах информационных единиц, допустимых в той конкретной СУБД, в среде которой проектируется БД. Этап создания ДЛМ называется даталогическим проектированием. Описание логической структуры БД на языке СУБД называется схемой.

Для привязки ДЛМ к среде хранения используется модель данных физического уровня, или физическая модель. Эта модель базы данных определяет используемые запомина-

ющие устройства, способы физической организации данных в среде хранения.

Физическая модель так же, как и ДЛМ, строится с учетом возможностей, предоставляемых СУБД. Описание физической структуры БД называется схемой хранения. Соответствующий этап проектирования БД называется физическим проектированием.

В некоторых СУБД, помимо описания общей логической структуры БД (т.е. схемы), имеется возможность описать логическую структуру БД с точки зрения конкретного пользователя. Такая модель называется внешней, а ее описание называется подсхемой (рис. 1.3).

Если СУБД поддерживает уровень подсхем, то перед проектировщиком встает задача определения подсхем. Это также можно рассматривать как этап проектирования БД.

Использование подсхем облегчает работу пользователя, так как он должен знать структуру не всей базы данных, а только той ее части, которая имеет непосредственное отношение к нему; кроме того, эта структура приспособлена к его потребностям.

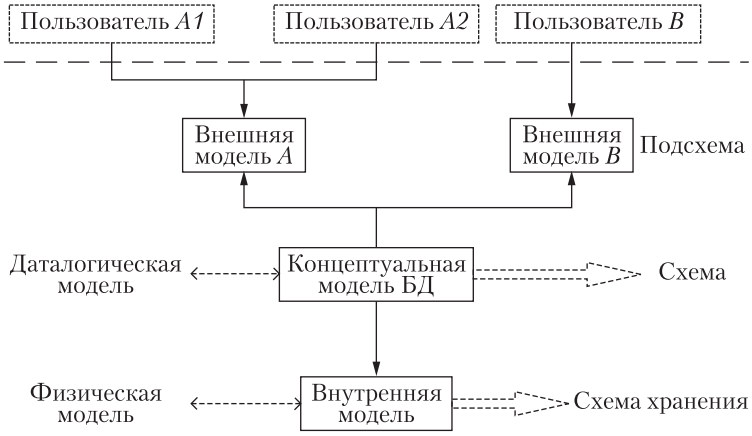


Рис. 1.3. **Модели и описания структуры БД, поддерживаемые СУБД**

Взаимосвязь этапов проектирования базы данных показана на рис. 1.4.

Первой строится инфологическая модель предметной области. Предварительная ИЛМ строится еще на предпро-

ектной стадии и затем уточняется на более поздних стадиях проектирования. Затем на ее основе строится даталогическая модель. Физическая и внешняя модели после этого могут строиться в любой последовательности, в том числе и параллельно. При проектировании БД возможен возврат на предыдущие этапы.

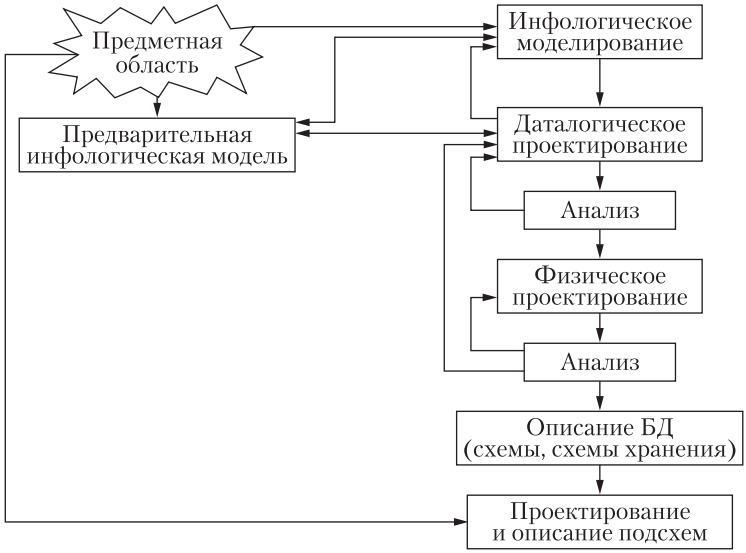


Рис. 1.4. **Взаимосвязь этапов проектирования**

Возможны два вида возвратов: первый обусловлен необходимостью пересмотра результата проектирования (например, для улучшения полученных характеристик, «обхода» ограничений и т.п.), второй вызван необходимостью уточнения предыдущей модели (обычно инфологической) с целью получения дополнительной информации для проектирования или при выявлении противоречий в модели.

1.4. Классификация баз данных

По форме представления информации различают видео- и аудиосистемы, а также системы мультимедиа. Эта классификация в основном показывает, в каком виде информация

из баз данных выдается пользователям: в виде изображения, звука или в виде сочетания разных форм отображения информации.

Наиболее распространенным пока являются базы данных, содержащих обычные символьные данные. Эти БД подразделяются на неструктурированные, частично структурированные и структурированные.

К неструктурированным БД относятся базы, организованные в виде семантических сетей. Частично структурированными можно считать БД, в которых информация представлена в виде обычного текста или гипертекста.

Структурированные БД по типу используемой модели делятся на иерархические, сетевые, реляционные, объектно-ориентированные, смешанные и мультимодельные. Наибольшее коммерческое использование в настоящее время имеют реляционные БД.

По типу хранимой информации БД делятся на документальные и фактографические. Документальные БД содержат сведения о документах на естественном языке — монографиях, научных отчетах, текстах законодательных актов и т.д. Фактографические БД содержат фактические сведения, например, данные о кадровом составе предприятия.

По характеру организации хранения данных и обращения к ним различают локальные (персональные), общие (интегрированные) и распределенные БД (рис. 1.5).

По условиям предоставления услуг различают бесплатные и платные БД. Платные БД делятся на бесприбыльные и коммерческие.

По форме собственности БД делятся на государственные и негосударственные.

По степени доступности различают общедоступные и с ограниченным кругом пользователей.

1.5. Классификация моделей данных

При работе с базами данных основными понятиями являются «данные» и «модель данных». Данные — это набор конкретных значений, параметров, характеризующих объект, условие, ситуацию или любые другие факторы [10]. Данные не обладают определенной структурой и становятся информаци-

ей тогда, когда пользователь задает им конкретную структуру, то есть вкладывает в них смысловое содержание.

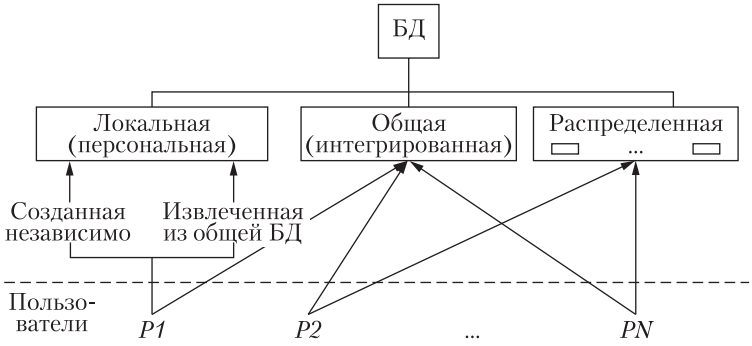


Рис. 1.5. **Классификация БД по характеру хранения данных и обращения к ним**

Модель данных — это некоторая абстракция, которая применительно к данным позволяет пользователям и разработчикам трактовать их уже как информацию, то есть сведения, содержащие не только данные, но и взаимосвязь между ними. Классификация моделей данных показана на рис. 1.6.

С учетом трехуровневой архитектуры баз данных (см. рис. 1.2) понятие модели данных трансформируется для каждого уровня. Например, физическая модель данных оперирует категориями, касающимися организации внешней памяти и структур хранения, используемых в данной операционной среде. В качестве физических моделей применяются различные методы размещения данных, основанные на файловых структурах и страничной организации данных.

Первостепенную важность имеют модели данных, используемые на концептуальном уровне. По отношению к ним внешние модели называются подсхемами и используют те же абстрактные категории, что и концептуальные модели данных.

При проектировании базы данных модель должна выражать информацию о предметной области в виде, независимом от используемой СУБД. Такая модель называется инфологической и отражает в естественной и удобной для разработчиков и других пользователей форме информационно-логический уровень абстрагирования, связанный с фиксацией и описанием объектов предметной области, их свойств и их взаимосвязей. Инфологические модели данных используются

на ранних стадиях проектирования для описания структур данных в процессе разработки приложения, а даталогические модели уже поддерживаются конкретной СУБД.

Документальные модели данных соответствуют представлению о слабоструктурированной информации, ориентированной в основном на свободные форматы документов, текстов на естественном языке.

Модели, ориентированные на формат документа, связаны прежде всего со стандартным общим языком разметки — SGML, который был утвержден ISO в качестве стандарта еще в 1980-х гг. Этот язык предназначен для создания других языков разметки, он определяет допустимый набор дескрипторов, их атрибуты и внутреннюю структуру документа. С помощью SGML можно описывать структурированные данные, организовывать информацию, содержащуюся в документах, представлять эту информацию в некотором стандартизованном формате.

Гораздо более простой и удобный, чем SGML, язык HTML позволяет определять оформление элементов документа и вносить специальные дескрипторы в документы, при помощи которых осуществляется процесс разметки. Дескрипторы на языке HTML в первую очередь предназначены для управления процессом вывода содержимого документа на экране с помощью программы-клиента (например, браузера) и определяют этим самым способ представления документа, но не его структуру. На языке HTML документ представляется набором элементов, причем начало каждого элемента, а в большинстве случаев и его конец, отмечается дескриптором, который называется тегом. В начале элемента указывается открывающий тег, а в конце — закрывающий. Например, элемент, соответствующий размечаемому документу, открывается тегом <html>, закрывается тегом </html> и содержит внутри себя элементы заголовка и тела документа, ограниченные специальными тегами <head> </head> и <body> </body>:

```
<html>
   <head>
      заголовок документа
   </head>
   <body>
      тело документа
   </body>
</html>
```

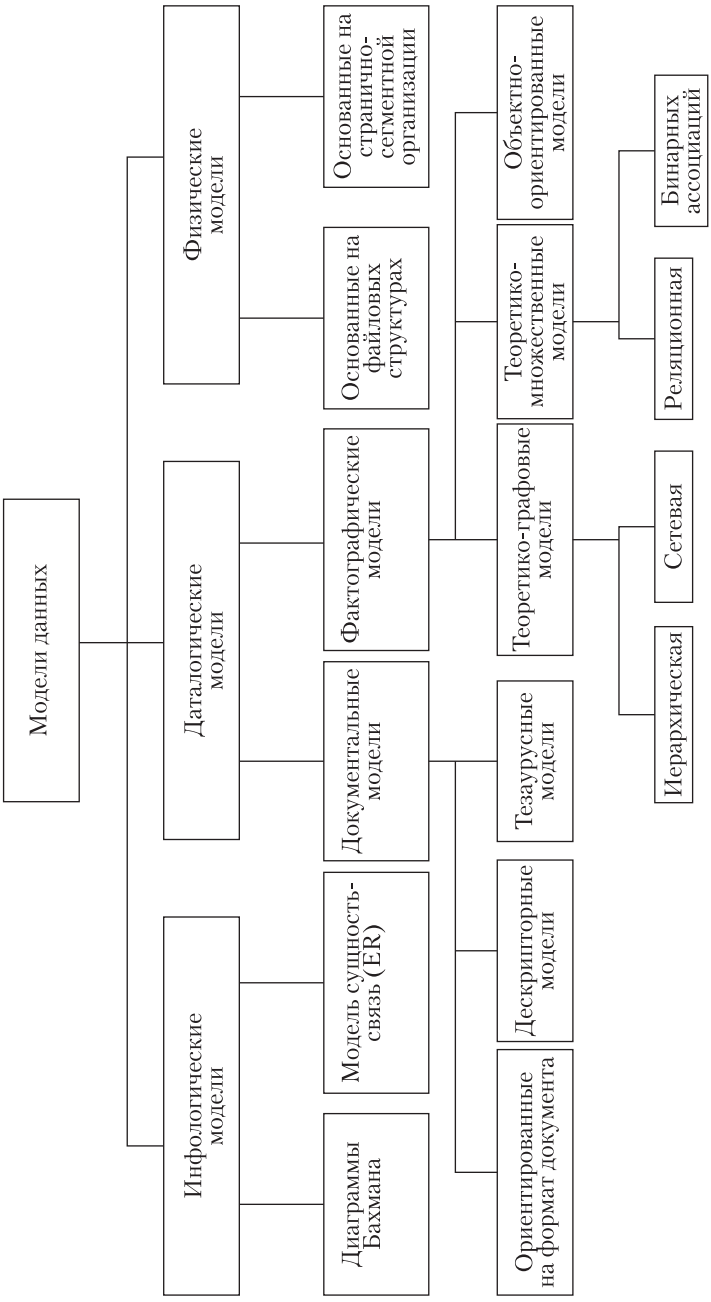


Рис. 1.6. **Классификация моделей данных**

В качестве компонента гипертекстовой базы данных, описываемой на языке HTML, используется текстовый файл, который может легко передаваться по сети с использованием протокола HTTP. Эта особенность, а также то, что HTML является открытым стандартом и огромное количество пользователей имеют возможность применять возможности этого языка для оформления своих документов, безусловно, повлияли на рост популярности HTML и сделали его главным средством представления информации в Интернете.

Однако HTML сегодня уже не удовлетворяет в полной мере требованиям, предъявляемым современными разработчиками к языкам подобного рода. На смену ему пришел новый язык гипертекстовой разметки, мощный, гибкий и удобный язык XML.

XML — это расширяемый язык разметки, описывающий целый класс объектов данных, называемых XML-документами. Он используется в качестве средства для описания грамматики других языков и проверки правильности составления документов. Сам по себе XML не содержит никаких тегов, предназначенных для разметки, но определяет порядок их создания.

Тезаурусные модели основаны на принципе организации словарей, содержат определенные языковые конструкции и принципы их взаимодействия в заданной грамматике. Эти модели эффективно используются в системах-переводчиках, особенно многоязыковых переводчиках. Принцип хранения информации в этих системах и подчиняется тезаурусным моделям.

Дескрипторные модели — самые простые из документальных моделей, они широко использовались на ранних стадиях использования документальных баз данных. В этих моделях каждому документу соответствовал дескриптор — описатель. Этот дескриптор имел жесткую структуру и описывал документ в соответствии с характеристиками, требуемыми для работы с документами в документальной базе данных.

Иерархическая, сетевая и реляционная модели отражают способ установления связей между данными. Иерархическая и сетевая модели предполагают наличие связей между данными, имеющими какой-либо общий признак.

На рис. 1.7 условно изображено распределение учебных курсов К1, К2, К3 между преподавателями П1, П2, П3. В ИМ связи между данными о преподавателях и читаемых ими курсах могут быть отражены в виде дерева, где возможны

только односторонние связи от старших вершин к младшим (рис. 1.8). Это облегчает быстрый доступ к необходимой информации, но только если запросы учитывают структуру дерева. Например, оперативно можно определить, какие курсы читает преподаватель П2. Запросы, не учитывающие структуру дерева (например, какие преподаватели читают курс К1), выполняются медленнее.

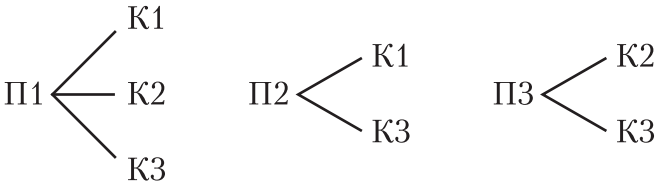


Рис. 1.7. **Распределение курсов между преподавателями**

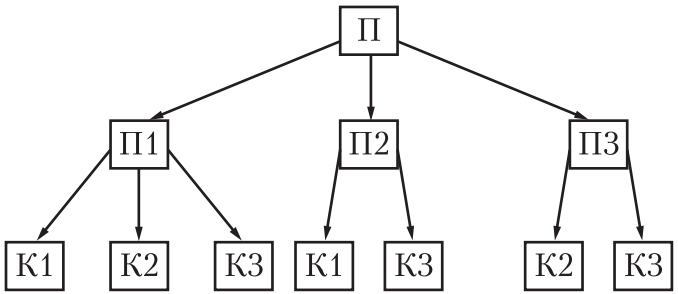


Рис. 1.8. **Иерархическая модель данных**

Указанный недостаток снят в СМ, где, по крайней мере теоретически, возможны связи «всех со всеми» (рис. 1.9). Использование ИМ и СМ ускоряет доступ к информации в БД. Но поскольку каждый элемент данных должен содержать ссылки на некоторые другие элементы, требуются значительные ресурсы как дисковой, так и основной памяти компьютера. Кроме того, для этих моделей характерна сложность реализации СУБД.

Реляционная модель является простейшей и наиболее привычной формой представления данных в виде таблицы (рис. 1.10). В теории множеств таблице соответствует термин «отношение» (relation), который и дал название реляционной

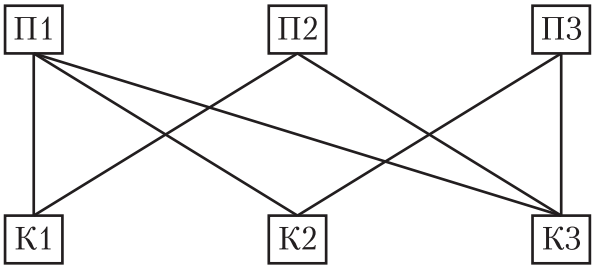


Рис. 1.9. **Сетевая модель данных**

модели. Достоинством РМ является сравнительная простота инструментальных средств ее поддержки, а недостатком — жесткость структуры данных и зависимость скорости выполнения операций от размера таблиц.

При создании моделей данных используются такие понятия, как «сущности», «атрибуты» и «связи». Сущность — это отдельный класс объектов предметной области (сотрудники или клиенты, понятия или события), который должен быть представлен в базе данных. Атрибут — это свойство, описывающее определенный аспект объекта, значение которого следует зафиксировать в описании предметной области. Связь является ассоциативным отношением между сущностями, при котором каждый экземпляр одной сущности соединен с некоторым

ПРЕП.

НП
П1
П2
П3

ЧИТАЕТ

НП	НК
П1	К1
П1	К2
П1	К3
П2	К1
П2	К3
П3	К2
П3	К3

КУРС

НК
К1
К2
К3

Рис. 1.10. **Реляционная модель данных:**
НП — номер преподавателя; НК — номер курса

(в том числе нулевым) количеством экземпляров другой сущности. Объектно-ориентированная модель расширяет определение сущности с целью включения в него не только атрибутов, которые описывают состояние объекта, но и действий, которые с ним связаны, т.е. его поведение. В таком случае говорят, что объект инкапсулирует состояние и поведение.

В настоящее время наиболее распространенными являются системы управления базами данными, поддерживающие реляционную модель данных. Эти системы называются реляционными СУБД.

1.6. Архитектура и типы СУБД

По своей архитектуре СУБД делятся на одно-, двух- и трехзвенные [19]. В однозвенной архитектуре (рис. 1.11, *а*) используется единственное звено (клиент), обеспечивающее необходимую логику управления данными и их визуализацию. В двухзвенной архитектуре (рис. 1.11, *б*) значительную часть логики управления данными реализует сервер баз данных (сервер БД), в то время как клиентское звено в основном занято отображением данных в удобном для пользователя виде. В трехзвенных СУБД (рис. 1.11, *в*) используется промежуточное звено — сервер приложений,

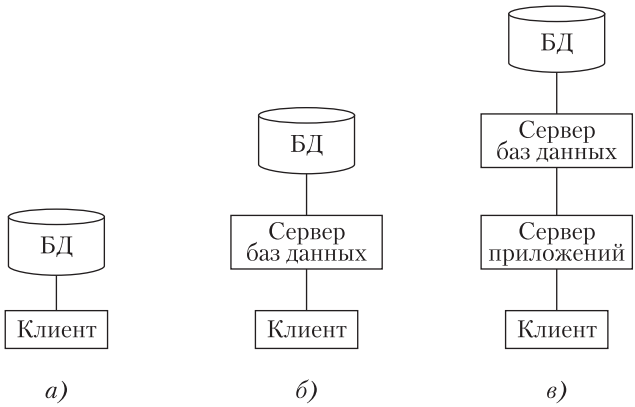


Рис. 1.11. **Архитектура СУБД:**
а — однозвенная; *б* — двухзвенная; *в* — трехзвенная

являющееся посредником между клиентом и сервером БД. Сервер приложений позволяет полностью избавить клиента от функций по управлению данными и обеспечению связи с сервером БД.

В зависимости от местоположения отдельных частей СУБД различают локальные и сетевые СУБД. Все части локальной СУБД размещаются на компьютере пользователя, обращающегося к базе данных. Чтобы с одной и той же БД одновременно могло работать несколько пользователей, каждый пользовательский компьютер должен иметь доступ к своей копии локальной БД. Существенной проблемой СУБД такого типа является синхронизация содержимого копий данных (репликация данных), именно поэтому для решения задач, требующих совместной работы нескольких пользователей, локальные СУБД не пригодны.

К сетевым относятся файл-серверные, клиент-серверные и распределенные СУБД. Непременным атрибутом этих систем является сеть, обеспечивающая аппаратную связь компьютеров и делающая возможной совместную работу множества пользователей с одной и той же базой данных.

В файл-серверных СУБД вся база данных обычно размещается на одном или нескольких запоминающих устройствах достаточно мощной машины, специально выделенной для этих целей и постоянно подключенной к сети. Такой компьютер называется файл-сервером. Безусловным достоинством СУБД этого типа является относительная простота ее создания и обслуживания, так как фактически все сводится лишь к организации локальной сети и установке на подключенных к ней компьютерах сетевых операционных систем. Между локальными и файл-серверными вариантами СУБД нет особых различий, так как в них все части СУБД сосредоточены на компьютере пользователя. По архитектуре они обычно являются однозвенными, но в некоторых случаях могут использовать сервер приложений. Недостатком файл-серверных систем является значительная нагрузка на сеть. Например, если пользователю, работающему на клиентском компьютере, нужно отыскать сведения об одной из книг, имеющихся в библиотеке, то по сети вначале передается весь файл, содержащий сведения обо всех книгах, и лишь затем в созданной таким образом локальной копии данных отыскиваются нужные сведения. При интенсивной работе с данными нескольких десятков пользователей пропускная способность сети может оказаться недостаточной,

и пользователя будут раздражать значительные задержки в реакции СУБД на его требования. Файл-серверные СУБД могут успешно использоваться в относительно небольших организациях с количеством клиентских мест до нескольких десятков.

Клиент-серверные (двухзвенные) системы значительно снижают нагрузку на сеть, так как клиент общается с данными через специализированного посредника — сервер БД, который размещается на машине с базой данными. Сервер БД принимает запрос от клиента, отыскивает в данных нужную запись и передает ее клиенту. Таким образом, по сети передаются относительно короткий запрос и единственная нужная запись, даже если база данных содержит сотни тысяч записей. Как правило, запрос к серверу формируется на специальном языке запросов SQL, поэтому часто серверы БД называются SQL-серверами. Серверы БД представляют собой относительно сложные программы, разрабатываемые различными фирмами, например: Microsoft SQL Server (SQL Server) производства корпорации Microsoft, Sybase Adaptive Server корпорации Sybase, Oracle производства одноименной корпорации, DB2 корпорации IBM, InterBase корпорации Borland и т.д. Клиент-серверные СУБД обеспечивают функционирование, или масштабируются, до сотен и тысяч клиентских мест.

Распределенные СУБД могут содержать несколько десятков и сотен серверов БД. Количество клиентских мест в них может достигать десятков и сотен тысяч. Обычно такие СУБД обеспечивают работу организаций государственного масштаба (например, Центральной избирательной комиссии РФ), отдельные подразделения которых рассредоточены на значительной территории. В распределенных СУБД некоторые серверы могут дублировать друг друга с целью достижения предельно малой вероятности отказов и сбоев, которые могут исказить жизненно важную информацию.

Актуальность распределенных СУБД возросла в связи со стремительным развитием Интернета. Опираясь на возможности Интернета, распределенные системы строят не только организации государственного масштаба, но и относительно небольшие коммерческие предприятия, обеспечивая своим сотрудникам работу с корпоративными данными на дому и в командировках.

1.7. Публикация данных в Интернете

Наиболее развитой частью Интернета является World Wide Web — Всемирная паутина, представляющая собой систему публикации ресурсов в виде гипертекстовых документов. Технологии, обеспечивающие работу WWW, называются Web-технологиями. Чтобы выделить аппаратные и программные компоненты, реализующие Web-технологии, серверный компьютер, предназначенный для размещения документов, удобно именовать WWW-сервером, а программу, находящуюся на WWW-сервере и обеспечивающую доступ к документам, — Web-сервером. В настоящее время наиболее распространенным является Web-сервер Apache, версии которого работают практически на всех известных аппаратно-программных платформах, в отличие от Web-сервера Microsoft Internet Information Services (IIS), ориентированного на операционные системы Windows.

Для просмотра гипертекстовых документов, называемых Web-страницами, используются специальные программы — браузеры, примерами которых являются Microsoft Internet Explorer, Firefox и Opera. Браузер интерпретирует команды, содержащиеся в гипертекстовом документе, и отображает этот документ на экране для просмотра пользователем.

Для автоматического создания Web-страниц с изменяющейся информацией применяются специальные расширения Web-сервера, называемые Web-приложениями. Типичная задача, выполняемая Web-приложением, — это получение из базы данных информации, которая затем помещается в Web-страницу и передается Web-серверу, пересылающему эту Web-страницу браузеру. В этом случае схема доступа к базе данных (рис. 1.12) соответствует трехуровневой архитектуре СУБД (см. рис. 1.11).

Использование баз данных при публикации информации в WWW существенно расширяет возможности Web-сервера

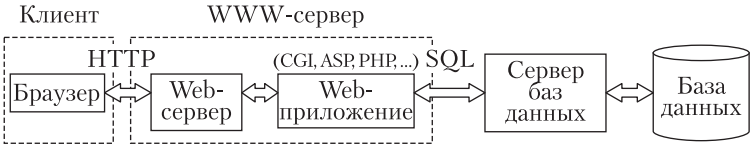


Рис. 1.12. **Схема доступа к базе данных через Интернет**

и решает многие проблемы, связанные с ограничением доступа к информации и эффективностью поиска необходимых данных.

В свою очередь, использование браузера в качестве клиентской программы позволяет получить доступ к базе данных с компьютера, оснащенного любой операционной системой, для которой имеется браузер. При этом не требуется разрабатывать специальные приложения, поскольку язык HTML одинаково интерпретируется браузерами, независимо от того, в какой операционной системе они функционируют. Кроме того, при изменении структуры базы данных не требуется обновлять программное обеспечение пользователей этой базы данных, так как модификация касается только той части программного обеспечения, которая находится на WWW-сервере и доступна после изменений всем, кто имеет право доступа к этому серверу.

Благодаря всем этим достоинствам доступ к базам данных на основе Web-технологии применяется и в локальных сетях. Сети, использующие Web-технологию для доступа к данным, называются интрасетями, или *интранетом* (intranet).

Для доступа к базам данных через Интернет наиболее часто используется один из двух подходов [7]:

1) однократное или периодическое преобразование содержимого базы данных в статические гипертекстовые документы. В этом случае база данных просматривается специальной программой, создающей множество связанных HTML-документов, содержащих информацию из базы данных. Полученные HTML-файлы размещаются на одном или нескольких WWW-серверах. Этот вариант достаточно эффективен при работе с небольшими, редко обновляемыми базами данных, имеющими простую структуру, а также при низких требованиях к актуальности данных, предоставляемых через Интернет;

2) динамическое создание гипертекстовых документов на основе информации, содержащейся в базе данных, и информации, переданной клиентом Web-серверу. В этом варианте доступ к базе данных обеспечивается специальным Web-приложением (CGI, ASP, PHP и т.п.), вызываемым Web-сервером в ответ на запрос, полученный от клиента. Web-приложение обрабатывает запрос, производит необходимую выборку из базы данных и на ее основе формирует выходной HTML-документ, возвращаемый клиенту. Такое решение эффективно для больших баз данных со сложной структурой. Данный вариант позволяет также обеспечить возможность изменения данных, хранящихся в базе, с учетом информации, поступающей от клиента.

Глава 2

РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ

2.1. Основные понятия

Реляционная модель данных базируется на теории множеств, в которой применяются специальные математические символы, используемые в тексте учебного пособия и поясняемые далее:

$\in$ — принадлежность элемента множеству (например, $d \in D$ означает, что элемент d принадлежит множеству D);

$\notin$ — отрицание принадлежности элемента множеству (например, $d \notin D$ означает, что элемент d не принадлежит множеству D);

$|D|$ — мощность множества D, равная числу элементов в этом множестве;

$\subseteq$ — вхождение одного множества в другое множество (например, $A \subseteq D$ означает, что все элементы множества A одновременно являются элементами множества D и $|A| \leq |D|$);

$\subset$ — включение одного множества в другое множество (например, $A \subset D$ означает, что все элементы множества A одновременно являются элементами множества D и $|A| < |D|$);

$\cup$ — объединение множеств (например, $X \cup Y$ означает множество, элементы которого принадлежат множеству X или множеству Y);

$\cap$ — пересечение множеств (например, $X \cap Y$ означает множество, элементы которого принадлежат одновременно множеству X и множеству Y);

$\varnothing$ — пустое множество, т.е. множество, в котором отсутствуют элементы;

$\{a, b, c\}$ — множество, состоящее из элементов a, b, c;

$\{e \mid y\}$ — множество элементов e, удовлетворяющих условию y; при записи условия могут использоваться символы $\wedge$ и $\vee$, обозначающие логические операторы «И» и «ИЛИ»

соответственно, например, $\{x \mid x \in R \wedge x \notin S\}$ обозначает множество элементов x, которые принадлежат множеству R и не принадлежит множеству S.

Реляционная БД — это совокупность отношений, содержащих всю информацию, которая должна храниться в БД.

Математически термин «отношение» определяется следующим образом.

Пусть даны N множеств D_1, D_2, ..., D_N. Отношением R над этими множествами называется множество упорядоченных N-кортежей вида $<d_1, d_2, ..., d_N>$, где $d_1 \in D_1$, ..., $d_N \in D_N$ ($N \geq 1$). Множества D_1, D_2, ..., D_N называются доменами (областями определения) отношения R.

Поясним это определение конкретным примером. Пусть даны четыре домена: D_1 — множество целых чисел, обозначающих номера преподавателей; D_2 — множество символьных строк, представляющих собой фамилии преподавателей; D_3 — множество символьных строк, представляющих собой названия должностей; D_4 — множество целых чисел, обозначающих стаж работы преподавателей. На рис. 2.1 показан пример отношения R, состоящего из пяти кортежей.

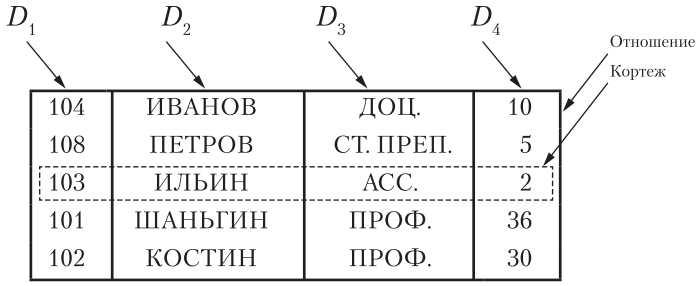


Рис. 2.1. **Отношение с математической точки зрения**

Каждый кортеж состоит из четырех элементов, которые выбираются каждый из своего домена. Порядок элементов в каждом кортеже строго определен: первый элемент кортежа выбирается из домена D_1, второй элемент — из домена D_2 и т.д. Каждый элемент кортежа представляет собой значение одного из атрибутов, соответствующего одному из доменов.

С программной точки зрения отношение является файлом (рис. 2.2), каждая запись в файле представляет собой кортеж отношения, а поля в записи содержат значения соответствующих атрибутов или доменов.

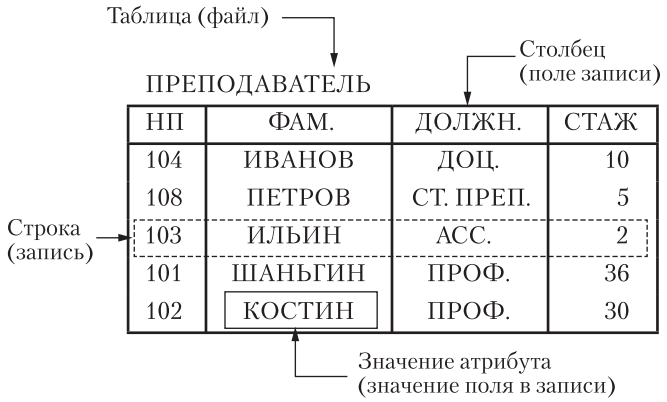


Рис. 2.2. **Отношение с точки зрения обработки данных**

Итак, разными точками зрения допускается следующая терминология:

Математически	Визуально	Программно
Отношение	Таблица	Файл
Кортеж	Строка	Запись в файле
Атрибут	Столбец	Поле записи

Количество атрибутов в кортеже, или число столбцов в таблице, называется *степенью отношения*. Текущее число кортежей, или строк, называется *мощностью отношения* и обозначается как $|R|$. Степень отношения не изменяется после создания отношения, но мощность отношения будет изменяться при добавлении новых и удалении старых кортежей. *Схемой отношения* R называется перечень атрибутов A_i данного отношения с указанием домена D_i, к которому они относятся:

$$S_R = (A_1, A_2, ..., A_N), \text{ где } A_i \subseteq D_i,\ 1 \le i \le N$$

По определению все кортежи различаются. Для однозначной идентификации конкретного кортежа используется так называемый первичный ключ отношения. *Первичный ключ* — это атрибут, или набор из минимального числа атрибутов, который однозначно идентифицирует конкретный кортеж и не содержит дополнительных атрибутов. Это значит, что если отдельный произвольный атрибут исключить из первичного ключа, то оставшихся атрибутов будет недостаточно для однозначной идентификации отдельных кортежей. Например,

в отношении ПРЕПОДАВАТЕЛЬ первичным ключом может быть только номер преподавателя НП, в таблице ЧИТАЕТ первичным ключом является набор атрибутов <НП, НК>, обозначающих номер преподавателя и номер читаемого учебного курса (рис. 2.3).

Для краткости отношение *R* принято обозначать именем, после которого в скобках перечисляются его атрибуты, причем атрибуты, входящие в первичный ключ указываются в начале списка и подчеркиваются, например:

R1(НП, ФАМ., ДОЛЖН., СТАЖ) *R2* (НП, НК)

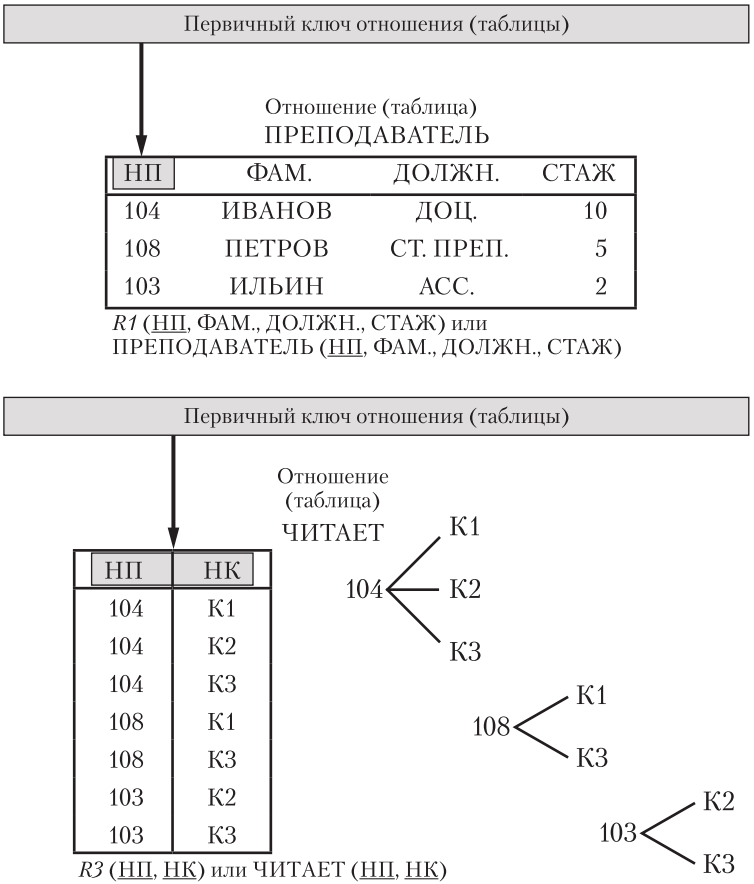


Рис. 2.3. **Первичные ключи**

В реляционной БД таблицы взаимосвязаны и соотносятся друг с другом как главные и подчиненные. Одной строке главной таблицы могут соответствовать несколько строк подчиненной таблицы (рис. 2.4). Связь главной и подчиненной таблиц осуществляется через первичный ключ (primary key) главной таблицы и внешний ключ (foreign key) подчиненной таблицы. Внешний ключ — это атрибут, или набор атрибутов, подчиненной таблицы, который в главной таблице является первичным ключом. Связь главной и подчиненной таблиц схематически изображается линией, соединяющей первичный и внешний ключи этих таблиц, с указанием одной стрелки со стороны главной таблицы и двух стрелок со стороны подчиненной таблицы (рис. 2.5).

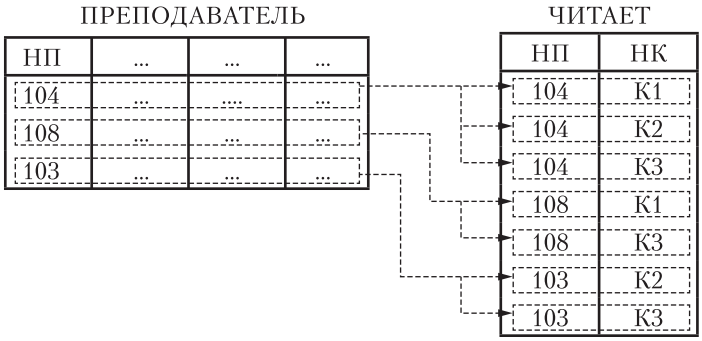


Рис. 2.4. **Связь главной таблицы ПРЕПОДАВАТЕЛЬ с подчиненной таблицей ЧИТАЕТ**

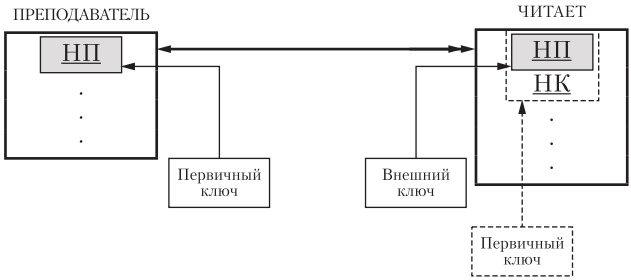


Рис. 2.5. **Схематическое изображение связи главной таблицы ПРЕПОДАВАТЕЛЬ с подчиненной таблицей ЧИТАЕТ**

Файл, в котором хранится отношение, может содержать довольно много записей, соответствующих кортежам. Для ускорения доступа к нужному кортежу файл индексируется. В качестве индексного ключа используется атрибут или набор атрибутов, определенный в отношении. В частности, индексным ключом может быть первичный ключ.

В результате индексирования создается дополнительный индексный файл, упорядоченный по значениям индексного ключа. Структура индексного файла может быть разной, но должна обеспечивать быстрый поиск. На рис. 2.6 показана возможная структура индексного файла, ускоряющего поиск в файле ПРЕПОДАВАТЕЛЬ по номеру преподавателя.

ПРЕП (индексный файл)

№ записи	Атрибут НП	№ записи в файле ПРЕПОДАВАТЕЛЬ
1	101	4
2	102	5
3	103	3
4	104	1
5	108	2

ПРЕПОДАВАТЕЛЬ

№ записи	НП	ФАМ.	ДОЛЖН.	СТАЖ
1	104	ИВАНОВ	ДОЦ.	10
2	108	ПЕТРОВ	СТ. ПРЕП.	5
3	103	ИЛЬИН	АСС.	2
4	101	ШАНЬГИН	ПРОФ.	36
5	102	КОСТИН	ПРОФ.	30

Рис. 2.6. **Пример индексного файла**

2.2. Реляционная алгебра

Реляционная алгебра служит математической основой реляционной ДЛМ. Так как отношение является множеством, то реляционная алгебра является алгеброй взаимосвязей между особыми множествами, называемыми отношениями.

В реляционной алгебре существует ряд операций над отношениями, например: проекция, выборка, соединение, объединение, пересечение, вычитание, умножение.

2.2.1. Проекция

Воспользуемся в качестве примера следующим отношением *R*, представленным в виде таблицы (рис. 2.7).

R *r*

НП	ЗавК	Нтел
102	Шаньгин	2854
104	Вернер	2882
108	Вернер	2882
125	Шаньгин	2854

Рис. 2.7. **Отношение *R* (*r* — кортеж)**

Пусть для конкретного кортежа *r*, являющегося элементом отношения *R*, *r*[*X*] обозначает расположенные в ряд составляющие кортежа, соответствующие множеству атрибутов *X*. Например, если *X*={ЗавК, Нтел}, то *r*[*X*] = <Вернер, 2882>.

Проекцией отношения *R* на *X* называют новое отношение

$$R[X] = \{ r[X] \mid r \in R \}.$$

Проекции *R*[ЗавК, Нтел] и *R*[НП, ЗавК] показаны на рис. 2.8.

R2 = *R*[ЗавК, Нтел]

ЗавК	Нтел
Шаньгин	2854
Вернер	2882

R1 = *R*[НП, ЗавК]

НП	ЗавК
102	Шаньгин
104	Вернер
108	Вернер
125	Шаньгин

Рис. 2.8. **Проекции *R*[ЗавК, Нтел] и *R*[НП, ЗавК]**

То есть при проецировании из исходного отношения *R* удаляется часть атрибутов, не указанных в *X*. При этом если в полученном таким способом отношении окажутся одинаковые кортежи, то из них оставляют по одному представителю. Следовательно, проекция является операцией, при которой из отношения выделяются только нужные столбцы.

2.2.2. Выборка

В противоположность проекции, которая выделяет из отношения нужные столбцы, выборкой называют операцию, при которой отношение исследуют по строкам и выделяют множество строк, удовлетворяющих заданным условиям.

Выборкой из отношения R по условию Θ для множества атрибутов X называют новое отношение

$$R[\,\Theta(X)\,] = \{\, r \mid r \in R \wedge \Theta(r[X])\,\}.$$

Так, выборка по условию $\Theta(r[X])$ = (ЗавК='Шаньгин'), т.е. информация о преподавателях, работающих на кафедре Шаньгина, показана на рис. 2.9.

R3 = *R* [ЗавК='Шаньгин']

НП	ЗавК	Нтел
102	Шаньгин	2854
125	Шаньгин	2854

Рис. 2.9. **Выборка**

2.2.3. Соединение

Операция соединения обратна операции проекции и предназначена для создания одного нового отношения из двух уже существующих отношений. Новое отношение получается путем конкатенации (сцепления) кортежей первого отношения R с кортежами второго отношения S. Только те кортежи подвергаются конкатенации, в которых значение заданного атрибута X в отношении R совпадает со значением заданного атрибута Y в отношении S:

$$R\,[X{=}Y]\,S = \{<r,s> \mid r \in R \wedge s \in S \wedge (r[X] = s[Y])\}.$$

Если R имеет N атрибутов (столбцов), а S — M атрибутов, то отношение $R[X{=}Y]S$ будет иметь $N + M$ атрибутов (столбцов). В получаемом отношении в двух столбцах всегда будут содержаться одинаковые значения. Если один из этих столбцов удалить, то результат принято называть естественным соединением.

Например, естественное соединение *R1* [ЗавК=ЗавК] *R2* отношений *R1* и *R2* (рис. 2.10), показанных на рис. 2.8, совпадает с отношением R, приведенным на рис 2.7.

2.2.4. Объединение

Объединение — это операция получения отношения, полностью объединяющего кортежи, содержащиеся в отношениях *R* и *S*. Множества атрибутов и порядок следования атрибутов в кортежах должны быть одинаковыми в отношениях *R* и *S*. Результирующее отношение называется множеством-суммой

$$R \cup S = \{ x \mid x \in R \vee x \in S \}.$$

R1

НП	ЗавК
102	Шаньгин
104	Вернер
108	Вернер
125	Шаньгин

R2

ЗавК	Нтел
Шаньгин	2854
Вернер	2882

X={ЗавК} *Y*={ЗавК}

R1 [ЗавК = ЗавК] *R2*

R

НП	ЗавК	ЗавК	Нтел
102	Шаньгин	Шаньгин	2854
104	Вернер	Вернер	2882
108	Вернер	Вернер	2882
125	Шаньгин	Шаньгин	2854

Естественное соединение

Рис. 2.10. **Соединение**

2.2.5. Пересечение

Результатом данной операции является отношение, состоящее из общих кортежей отношений *R* и *S*:

$$R \cap S = \{ x \mid x \in R \wedge x \in S \}.$$

2.2.6. Вычитание

Это операция получения отношения, состоящего из кортежей, которые являются кортежами отношения *R* и не являются кортежами отношения *S*:

$$R \setminus S = \{ x \mid x \in R \wedge x \notin S \}.$$

2.2.7. Умножение

Результатом данной операции является декартово, или прямое, произведение: $R \otimes S = \{ <r, s> \mid r \in R \wedge s \in S \}$.

Если умножаются отношение R степени m и отношение S степени n, то получается отношение $R \otimes S$ степени $(m+n)$.

Мощность отношения $R \otimes S$ равна $i \cdot j$, где i и j — мощности отношений соответственно R и S.

Например, если

$R =$

G2
A
B

, $S =$

G1	F1
d	3
h	7

,

то $R \otimes S =$

G2	G1	F1
A	d	3
A	h	7
B	d	3
B	h	7

С помощью рассмотренных операций реляционной алгебры можно формулировать запросы к БД для получения необходимой информации, а проекция используется также для нормализации отношений, которая проводится с целью удаления избыточных данных из таблиц в БД.

2.3. Примеры запросов на языке реляционной алгебры

Реляционную алгебру и предусмотренные в ней операции можно использовать в качестве языка для формулировки запросов к базе данных. Примеры такого использования рассмотрим для базы данных (рис. 2.11), в пяти таблицах которой хранятся сведения о жителях, их квартирах, доходах и телефонах:

```
PERSON   (Nom, FIO, Rdate, Pol, SumD, Adr)
FLAT     (Adr, Skv, Nrooms, KCategory)
TPHONE   (Ntel, TCategory, Adr)
PROFIT   (Id, Source, Money)
HAVE_D   (Nom, Id)
```

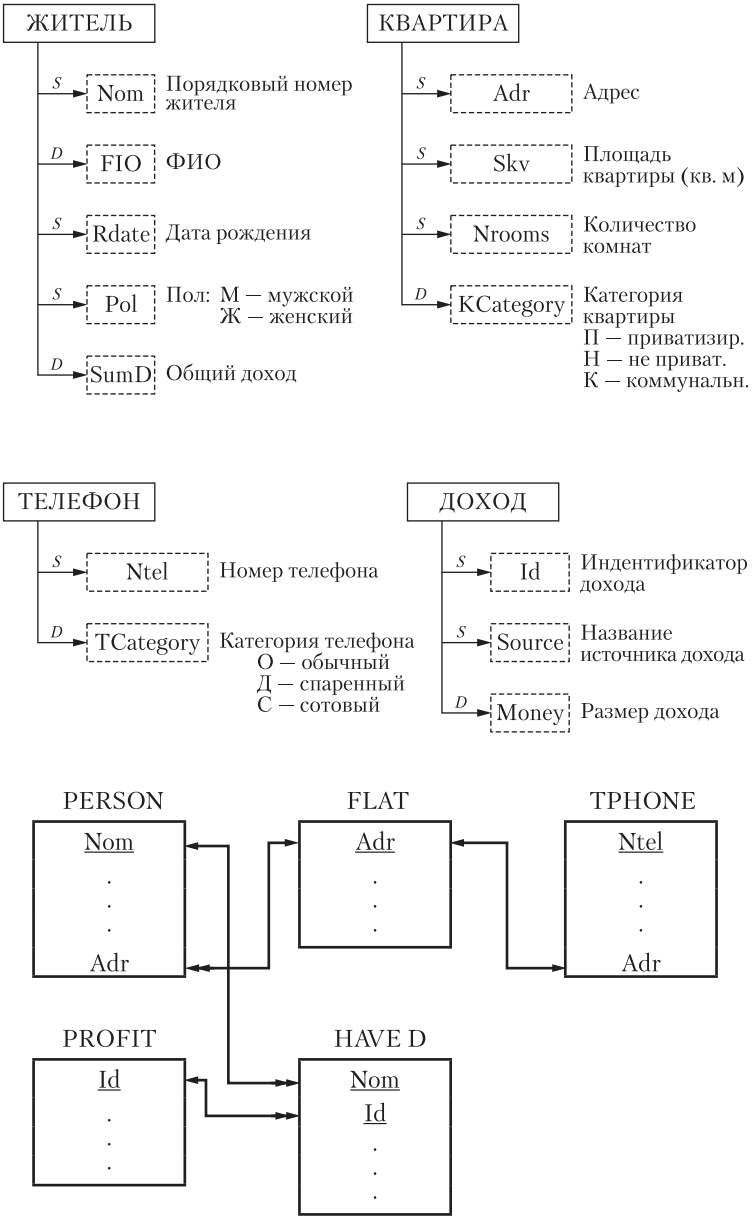


Рис. 2.11. **Учебная база данных**

Год рождения жителя с номером 199 определяется запросом

```
(person[nom=199])[Rdate]
```

Сведения о жителях, проживающих в квартире с адресом 901-15, можно получить по запросу

```
person[Adr = '901-15']
```

Все сведения о жителе с номером 199 определяются запросом

```
(((((PERSON[Nom=199])[Adr=Adr]FLAT)[Adr=Adr]
      TPHONE)[Nom=Nom]HAVE_D)[Id=Id]PROFIT)
```

Список дат рождения и адресов для всех жителей формируется запросом

```
person[Rdate, Adr]
```

Определить для каждого жителя источник дохода, дающий больше 500 рублей, можно с помощью запроса

```
(((PERSON[Nom=Nom]Have_D)[Id=Id]PROFIT)[Money>500])
[Nom, Source]
```

Язык реляционной алгебры не является «дружественным», поскольку выражения реляционной алгебры не всегда понятны и привычны обычному пользователю.

Как правило, реляционная алгебра используется для внутреннего представления запросов, а для взаимодействия с БД были разработаны языки запросов, ориентированные на пользователя, например, QBE и SQL. Наиболее популярным языком является SQL, ставший стандартом для реляционных СУБД.

Глава 3

ЯЗЫКИ БАЗ ДАННЫХ

Язык для работы с данными, который способна воспринимать СУБД, состоит из двух частей: *языка определения данных* (Data Definition Language — DDL) и *языка манипулирования данными* (Data Manipulation Language — DML). Язык DDL используется для определения схемы базы данных, а язык DML — для чтения и обновления данных, хранимых в базе. Эти языки называются *подъязыками данных* [13], поскольку в них отсутствуют такие конструкции для выполнения вычислительных операций, обычно используемых в языках программирования высокого уровня, как условные операторы или операторы цикла.

Во многих СУБД предусмотрена возможность включения операторов подъязыка данных в программы, написанные на таких языках программирования высокого уровня, как Pascal, C, C++, Java или Visual Basic. В этом случае язык высокого уровня принято называть *базовым включающим языком* (host language). Перед компиляцией программы на базовом языке, содержащей включенные операторы подъязыка данных, такие операторы удаляются и заменяются вызовами функций. Затем эта предварительно обработанная программа обычным образом компилируется с помещением результатов в объектный модуль, который компонуется с библиотекой, содержащей вызываемые в программе функции СУБД. После этого полученный программный код готов к выполнению.

Помимо возможности включения операторов подъязыка данных в программы, для большинства подъязыков данных предусмотрены также средства интерактивного выполнения операторов, вводимых пользователем непосредственно с компьютера в диалоговом режиме.

3.1. Язык определения данных (DDL)

Этот описательный язык позволяет администратору баз данных или пользователю задавать и именовать сущности и атрибуты, необходимые для работы некоторого приложения, а также связи, имеющиеся между различными сущностями, указывая, кроме того, ограничения целостности и защиты.

Схема базы данных состоит из набора определений, выраженных на языке определения данных. Язык DDL используется как для определения новой схемы, так и для модификации уже существующей.

Результатом компиляции DDL-операторов является набор таблиц, хранимый в особых файлах, называемых *системным каталогом*. В системном каталоге интегрированы метаданные, т.е. данные, которые описывают объекты базы данных, а также позволяют упростить способ доступа к ним и управления ими. Метаданные включают определения записей, элементов данных, а также других объектов, представляющих интерес для пользователей или необходимых для работы СУБД. Перед доступом к реальным данным СУБД обычно обращается к системному каталогу. Синонимами термина «системный каталог» являются «словарь данных» и «каталог данных».

3.2. Язык манипулирования данными (DML)

Этот язык включает в себя набор операторов для поддержки основных операций над данными, содержащимися в базе. К операциям манипулирования данными относятся:

— вставка в базу данных новых сведений;
— модификация сведений, хранимых в базе данных;
— извлечение сведений, содержащихся в базе данных;
— удаление сведений из базы данных.

Таким образом, одна из основных функций СУБД заключается в поддержке языка манипулирования данными, с помощью которого пользователь может создавать выражения для выполнения перечисленных выше операций с данными.

Понятие манипулирования данными применимо как к внешнему и концептуальному уровням, так и к внутреннему уровню. Однако на внутреннем уровне для этого приходится определять очень сложные низкоуровневые про-

цедуры, позволяющие выполнять доступ к данным с высокой эффективностью. На более высоких уровнях, наоборот, упор делается на большую простоту использования, и основные усилия связаны с обеспечением эффективного взаимодействия пользователя с системой.

Языки DML имеют разные базовые конструкции извлечения данных. Существуют два типа языков DML: *процедурный* и *непроцедурный*. Основное различие между ними заключается в том, что процедурные языки указывают то, *как* можно получить необходимый результат, тогда как непроцедурные языки описывают то, *какой* результат требуется получить. Как правило, в процедурных языках записи рассматриваются по отдельности, тогда как непроцедурные языки оперируют с целыми наборами записей.

Часть непроцедурного языка DML, которая отвечает за извлечение данных, называется *языком запросов данных* (Data Query Language — DQL). Язык запросов данных можно определить как высокоуровневый узкоспециализированный язык, предназначенный для удовлетворения различных требований по выборке информации из базы данных. В этом смысле термин «запрос» больше подходит для обозначения оператора извлечения данных, выраженного с помощью языка запросов данных.

Процедурный язык DML сообщает системе о том, какие данные необходимы, и точно указывает, как их можно извлечь. С помощью процедурного языка DML пользователь, а точнее — программист, указывает, какие данные ему необходимы и как их можно получить. Это значит, что пользователь должен определить все операции доступа к данным (осуществляемые посредством вызова соответствующих процедур), которые должны быть выполнены для получения требуемой информации. Обычно такой процедурный язык DML позволяет извлечь запись, обработать ее и, в зависимости от полученных результатов, извлечь другую запись, которая должна быть подвергнута аналогичной обработке, и т.д. Подобный процесс извлечения данных продолжается до тех пор, пока не будут извлечены все запрашиваемые данные. Обычно операторы процедурного языка DML встраиваются в программу на языке программирования высокого уровня, которая содержит конструкции для обеспечения циклической обработки и перехода к другим участкам кода. Языки DML сетевых и иерархических СУБД обычно являются процедурными.

Непроцедурный язык DML позволяет указать лишь то, какие данные требуются, но не то, как их следует извлекать. Непроцедурные языки DML задают весь набор требуемых данных с помощью одного оператора выборки или обновления. С помощью непроцедурных языков DML пользователь указывает, какие данные ему нужны, без определения способа их получения. СУБД транслирует выражение на языке DML в процедуру (или набор процедур), которая обеспечивает манипулирование указанным набором записей. Такой подход освобождает пользователя от необходимости знать подробности внутренней реализации структур данных и особенности алгоритмов, используемых для извлечения и возможного преобразования данных. В результате работа пользователя становится в определенной степени независимой от данных. Непроцедурные языки часто также называют *декларативными языками*. Реляционные СУБД в той или иной форме обеспечивают поддержку непроцедурных языков манипулирования данными, примерами которых являются структурированный язык запросов SQL и язык запросов по образцу QBE. Непроцедурные языки обычно проще понять и использовать, чем процедурные языки, поскольку пользователю достается меньшая часть работы, а СУБД — большая.

3.3. Генераторы

Генераторы относятся к *языкам четвертого поколения* (4GL). Если для организации некоторой операции с данными на языке третьего поколения (3GL) типа C++ потребуется написать сотни строк кода, то для реализации этой же операции на языке четвертого поколения достаточно 10—20 строк. В то время как языки третьего поколения являются процедурными, языки 4GL выступают как непроцедурные, поскольку пользователь определяет, *что* должно быть сделано, но не сообщает, *как* именно должен быть получен желаемый результат. Реализация языков четвертого поколения в значительной мере основана на использовании компонентов высокого уровня. Пользователю не требуется определять все этапы выполнения программы, необходимые для решения поставленной задачи, а достаточно лишь задать нужные параметры, на основании которых упомянутые выше компоненты автоматически осуществят генерацию приложения. В ряде случаев языки чет-

вертого поколения позволяют повысить производительность труда на порядок, но за счет ограничения типов задач, которые можно будет решать с их помощью. Выделяют следующие типы языков четвертого поколения [13]:

— языки представления информации, например языки запросов или генераторы отчетов;

— специализированные языки, например языки электронных таблиц и баз данных;

— генераторы приложений, которые при создании приложений обеспечивают определение, вставку, обновление или извлечение сведений из базы данных;

— языки очень высокого уровня, предназначенные для генерации кода приложений.

В качестве примеров языков четвертого поколения для работы с базами данных можно указать упоминавшиеся выше языки SQL и QBE, а также генераторы форм, генераторы отчетов, генераторы графического представления данных, генераторы приложений.

Генератор форм представляет собой интерактивный инструмент, предназначенный для быстрого создания шаблонов ввода и отображения данных в экранных формах. Генератор форм позволяет пользователю определить внешний вид экранной формы, ее содержимое и место расположения на экране. С его помощью можно задавать цвета элементов экрана, а также другие характеристики. Более совершенные генераторы форм позволяют создавать вычисляемые атрибуты с использованием арифметических операторов или агрегирующих функций, а также задавать правила проверки вводимых данных.

Генератор отчетов является инструментом создания отчетов на основе хранимой в базе данных информации. Он подобен языку запросов в том смысле, что пользователю предоставляются средства создания запросов к базе данных и извлечения из нее информации, используемой для представления в отчете. Однако генераторы отчетов, как правило, предусматривают большие возможности управления внешним видом отчета. Генератор отчета позволяет либо автоматически определять вид получаемых результатов, либо с помощью специальных команд создавать свой собственный вариант внешнего вида печатаемого документа.

Существуют два основных типа генераторов отчетов: языковой и визуальный. В первом случае для определения нужных для отчета данных и внешнего вида документа следует

ввести соответствующую команду на некотором подъязыке. Во втором случае для этих целей используется визуальный инструмент, подобный генератору форм.

Генератор графического представления данных — инструмент, предназначенный для извлечения информации из базы данных и отображения ее в виде диаграмм с графическим представлением существующих тенденций и связей. Обычно с помощью подобного генератора создаются гистограммы, круговые, столбчатые, точечные диаграммы и т.д.

Генератор приложений представляет собой инструмент для создания программ, взаимодействующих с базой данных. Применяя генератор приложений, можно сократить время, необходимое для проектирования полного объема требуемого прикладного программного обеспечения. Генераторы приложений обычно состоят из предварительно созданных модулей, содержащих фундаментальные функции, которые требуются для работы большинства программ. Эти модули, обычно создаваемые на языках высокого уровня, образуют «библиотеку» доступных функций. Пользователь указывает, *какие* задачи программа должна выполнить, а генератор приложений определяет, *как* их следует выполнить в генерируемой программе. Примером генератора приложений является мастер форм баз данных, имеющийся в системе визуального программирования Delphi.

3.4. Структурированный язык запросов SQL

3.4.1. Стандарты и разновидности языка SQL

Структурированный язык запросов SQL (Structured Query Language) предназначен для обработки и чтения информации, хранящейся в компьютерной базе данных [4]. SQL — это язык программирования, который применяется для организации взаимодействия пользователя с базой данных.

SQL реализован в более чем сотне СУБД, работающих как на персональных, так и больших компьютерах, называемых мэйнфреймами (mainframe). Существует международный стандарт на этот язык.

Работа над официальным стандартом была инициирована Американским институтом национальных стандартов

(ANSI) в 1982 г. В 1986 г. стандарт был официально утвержден как стандарт ANSI, а в 1987 г. — в качестве стандарта Международной организации по стандартам (ISO). Этот стандарт, незначительно пересмотренный в 1989 г., обычно называют стандартом SQL-89, SQL1, или стандартом ANSI/ISO SQL-89.

Следующий стандарт, называемый SQL2, или SQL-92, был принят в ANSI в 1992 г. Незадолго до завершения работ по определению стандарта SQL2 была начата разработка стандарта SQL3. Первоначально планировалось завершить проект в 1995 г. и включить в язык некоторые объектные возможности: определяемые пользователями типы данных, поддержку триггеров и т.д. Реально работу над новым стандартом удалось частично завершить только в 1999 г., и по этой причине стандарт получил название SQL:1999.

В конце 2003 г. был принят и опубликован новый вариант международного стандарта SQL:2003. Вопреки ожиданиям специалистов, что в варианте стандарта, следующем за SQL:1999, будут всего лишь исправлены неточности SQL:1999, в SQL:2003 был специфицирован ряд новых и важных элементов языка.

Во всех указанных стандартах жестко определяется основной набор операторов и возможностей языка. Каждая реализация языка учитывает требования стандарта и может иметь дополнительные возможности, которых нет в стандарте.

Язык SQL можно использовать для доступа к базам данных в двух режимах: при интерактивной работе (т.е. в диалоговом режиме) и в прикладных программах. Соответственно имеются две разновидности языка: интерактивный SQL и программный SQL, которые по своим возможностям отличаются незначительно.

Есть два основных способа применения программного SQL:

1) *встроенный SQL*; при таком подходе оператор языка SQL либо записывается непосредственно в тексте программы, составляемой на обычном языке программирования, — это *статический SQL*, либо формируется программой в специальной области памяти — это *динамический SQL*; затем записанный или сформированный оператор выполняется в ходе исполнения программы;

2) *SQL, использующий интерфейс прикладного программирования* (API); при этом способе программа взаимодействует с СУБД посредством совокупности функций (API-функций).

Вызывая API-функции, программа передает в СУБД операторы SQL и получает обратно результаты запросов.

3.4.2. Основные элементы языка SQL

Операторы

В языке SQL имеется приблизительно 30 операторов. Каждый оператор заставляет СУБД выполнить определенное действие, например прочитать данные, создать таблицу или добавить новые данные в таблицу. Все операторы SQL имеют одинаковую структуру (рис. 3.1).

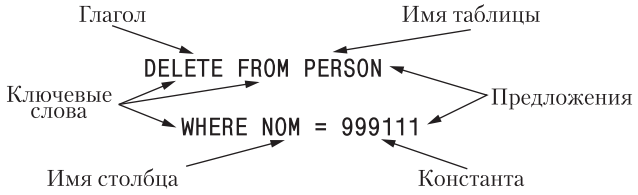


Рис. 3.1. **Структура оператора SQL**

Каждый оператор начинается с глагола, т.е. ключевого слова, описывающего действие, выполняемое оператором, например: DELETE (удалить), CREATE (создать), INSERT(добавить), UPDATE (изменить).

После глагола идет одно или несколько предложений или фраз. Предложение (фраза) описывает данные, с которыми работает оператор, или содержит уточняющую информацию о действии, выполняемом оператором. Каждое предложение также начинается с ключевого слова, такого, как WHERE (где), FROM (из), HAVING (имеющий) и т.п.

Одни предложения в операторе являются обязательными, а другие — нет.

В стандарте ANSI/ISO SQL-89 определены ключевые слова, которые применяются в качестве глаголов и в предложениях операторов. В соответствии со стандартом эти ключевые слова нельзя использовать для именования таких объектов БД, как таблицы, столбцы и пользователи.

Правила записи операторов SQL будем описывать с помощью синтаксических диаграмм, одна из которых показана на рис. 3.2.

Рис. 3.2. **Синтаксическая диаграмма для оператора DELETE**

Чтобы написать правильный оператор SQL или предложение, необходимо пройти по синтаксической диаграмме вдоль линий до точки, служащей концом диаграммы. На диаграмме, показанной на рис. 3.2, ключевые слова записаны прописными буквами. Однако почти во всех реализациях языка SQL ключевые слова можно набирать и прописными, и строчными буквами. Изменяемые элементы оператора на диаграмме представлены строчными буквами.

Имена

У каждого объекта в БД есть уникальное имя. Имена используют в операторах SQL и указывают, над каким объектом БД оператор должен выполнять действие. В соответствии со стандартом ANSI/ISO в SQL имена должны содержать от 1 до 18 символов, начинаться с латинской буквы и не содержать пробелов или специальных символов пунктуации. В стандарте SQL2 максимальное число символов в имени увеличено до 128.

Если в операторе указано имя таблицы, то предполагается, что происходит обращение к одной из таблиц, которую создали вы или которую вы считаете своей. Имея соответствующее разрешение, можно обращаться к таблицам, владельцами которых являются другие пользователи, с помощью полного имени таблицы. Полное имя таблицы состоит из имени владельца таблицы и собственного ее имени, разделенных точкой, например: ANTON.PERSON.

Если в операторе задается имя столбца, то СУБД сама определяет, в какой из таблиц, указанных в этом же операторе, содержится данный столбец. Однако если в операторе требуется указать два столбца из различных таблиц, но с одинаковыми именами, то в этом случае необходимо задать полные имена столбцов, которые точно определяют местонахождение столбцов.

Полное имя столбца состоит из имени таблицы, содержащей столбец, и имени столбца (простого имени), разделенных точкой. Например, полное имя столбца ADR в таблице PERSON имеет такой вид

```
PERSON.ADR или ANTON.PERSON.ADR
```

в том случае, если столбец находится в таблице, принадлежащей другому пользователю с именем ANTON.

Типы данных

В стандарте ANSI/ISO (SQL1) определены типы данных, которые можно использовать для представления информации в реляционной БД. Эти типы образуют лишь минимальный набор и поддерживаются во всех коммерческих СУБД:

1) строки символов постоянной длины;
2) целые числа;
3) масштабируемые числа. В столбцах этого типа хранятся числа, имеющие дробную часть и которые необходимо вычислять точно, например денежные суммы и проценты;
4) числа с плавающей запятой.

В стандарт SQL2 вошли еще несколько типов данных:

1) строки символов переменной длины;
2) дата и время;
3) поток байтов для хранения графических и видеоизображений, программных кодов и других неструктурированных данных;
4) строки национальных символов.

В большинстве коммерческих СУБД помимо типов данных, определенных в стандартах, имеется множество дополнительных типов данных. Например, в системе Delphi реализован так называемый локальный SQL для работы с таблицами, доступ к которым обеспечивают СУБД Paradox и dBase. В этой реализации, в частности, предусмотрены типы данных, перечисленные в табл. 3.1.

Константы

В стандарте ANSI/ISO (SQL1) определен формат числовых и строковых констант, или литералов, которые представляют конкретные значения данных.

Целые и десятичные константы, называемые точными числовыми литералами, в операторах SQL представляются в виде обычных десятичных чисел со знаком или без знака:

21 –375 2000 0.0 –4.579

Таблица 3.1

Соответствие типов данных

Тип SQL	Соответствует		Примечания
	Paradox	dBase	
SMALLINT	Short	Number(6, 0)	
INTEGER	Long Integer	Number(11, 0)	
DECIMAL(*x*, *y*)	BCD	—	
NUMERIC(*x*, *y*)	Number	Number(*x*, *y*)	
FLOAT(*x*, *y*) CHAR(*n*)	Number Alpha	Float(*x*, *y*) Character	
VARCHAR(*n*)	Alpha	Character	
DATE	Date	Date	
BOOLEAN	Logical	Logical	Булевы данные (TRUE/FALSE)
BLOB(*n*, *m*)	Binary Graphic	Binary —	Двоичные данные большого размера
TIME TIMESTAMP	Time Timestamp	— —	Дата и время
MONEY	Money	Float(20, 4)	Денежные величины
AUTOINC	Autoincrement	—	Автоинкрементный тип
BYTES(*n*)	Bytes	—	Двоичные данные

Примечание: *x* — точность (значащие цифры); *y* — масштаб (дробная часть); *n* — длина в байтах (символах); *m* — подтип BLOB-поля

Константы с плавающей запятой, называемые приблизительными числовыми литералами, задаются с помощью символа E, означающего умножение на 10 в степени:

$$1.5E3 \rightarrow 1{,}5 \cdot 10^3 \qquad -314E-2 \rightarrow -3{,}14.$$

Строковые константы заключаются в апострофы (одинарные кавычки):

'ИВАНОВ' 'Зеленоград, 801-101'.

Сам апостроф в составе строковой константы представляется двумя идущими подряд апострофами: 'ОБ''ЕКТ' → ОБ'ЕКТ

Календарные даты и время представляются в виде строковых констант. Форматы этих констант в различных СУБД отличаются друг от друга. Кроме того, способы записи времени и даты зависят от страны. В системе Delphi утилита конфигурации BDE позволяет задать необходимый формат этих констант:

'5/22/1999' '10:35:15'.

Выражения

В соответствии со стандартом ANSI/ISO (SQL1) в выражениях можно использовать четыре арифметические операции: сложение ($X + Y$), вычитание ($X - Y$), умножение ($X \cdot Y$) и деление (X/Y). Для формирования сложных выражений можно использовать круглые скобки.

Преобразование целых чисел в масштабируемые десятичные числа и десятичных чисел в числа с плавающей запятой согласно стандарту должно происходить автоматически. Таким образом, в одном числовом выражении можно использовать числовые данные разных типов.

Отсутствующие, неподходящие или неизвестные данные

Поскольку БД является моделью реального мира, то отдельные элементы данных в ней в какой-то конкретный момент времени будут либо отсутствовать, либо не подходить для характеристики объекта, либо иметь неизвестные значения. Такое значение в SQL обозначается величиной NULL. Например, в столбце SUMD таблицы PERSON содержатся величины общих доходов для каждого жителя. Для нового жителя, который только что зарегистрирован, общий доход неизвестен, поэтому для него в качестве значения в столбце SUMD следует поставить значение NULL.

3.4.3. Использование SQL для выборки (чтения) данных

Язык SQL предназначен в первую очередь для выполнения запросов. Для построения SQL-запросов, задающих выборку данных, используется оператор SELECT, который является наиболее функциональным из всех операторов SQL.

Оператор SELECT читает данные из БД и возвращает их в виде таблицы результатов запроса (рис. 3.3).

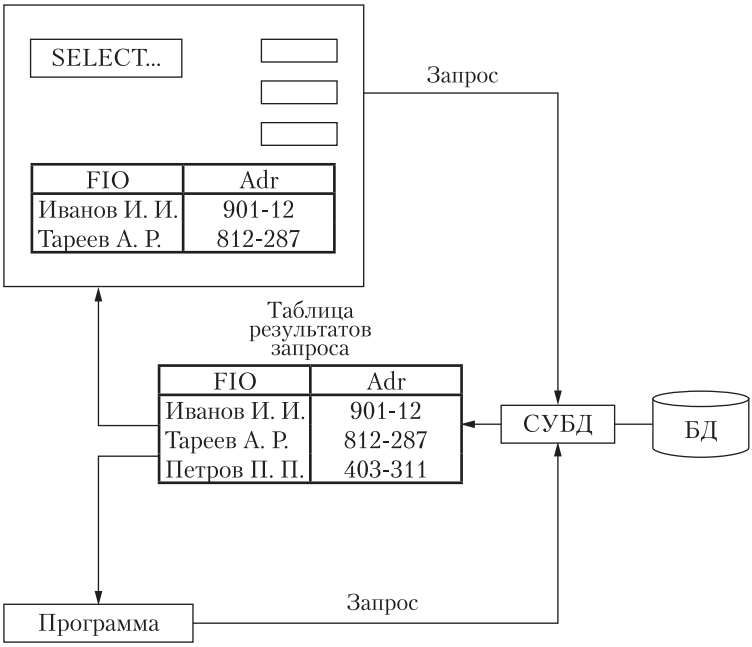


Рис. 3.3. **Выполнение SQL-запроса на чтение**

Оператор SELECT состоит из шести предложений, или фраз (рис. 3.4). Предложения SELECT и FROM являются обязательными. Четыре остальные включаются в оператор по мере необходимости. Функции каждого из предложений следующие:

1) в предложении SELECT указывается список возвращаемых столбцов, которые должны быть включены в таблицу результатов запроса. Возвращаемые столбцы могут содержать значения, считываемые из столбов таблиц базы данных, или значения, вычисляемые во время выполнения запроса;

2) в предложении FROM указывается список таблиц, которые содержат элементы данных, считываемые запросом;

3) предложение WHERE показывает, что в таблицу результатов запроса следует включать только строки, удовлетворяющие условию поиска;

4) предложение GROUP BY позволяет задавать итоговый запрос. Обычный запрос включает в таблицу результатов по одной строке для каждой строки из БД. Итоговый запрос

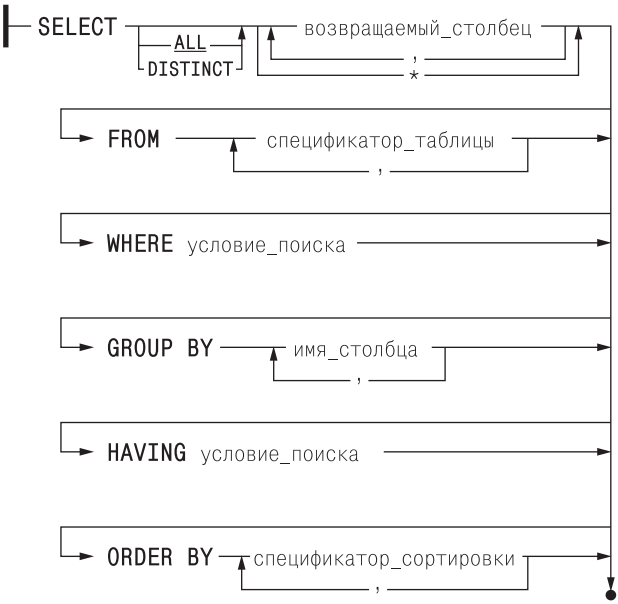


Рис. 3.4. **Синтаксическая диаграмма оператора SELECT**

вначале группирует строки БД по определенному признаку, а затем включает в таблицу результатов одну итоговую строку для каждой группы;

5) предложение HAVING показывает, что в таблицу результатов запроса следует включать только некоторые из групп, созданных с помощью предложения GROUP BY. Для отбора включаемых групп используется условие поиска;

6) предложение ORDER BY сортирует таблицу результатов запроса на основании данных, содержащихся в одном или нескольких столбцах.

В предложении SELECT, с которого начинаются все операторы SELECT, в списке возвращаемых столбцов следует указать такие элементы, как имя столбца, константа или выражение, которые будут возвращены в результате выполнения запроса. Для каждого элемента из этого списка в таблице результатов запроса будет создан один столбец.

Имя столбца идентифицирует один из столбцов, содержащихся в таблицах, которые перечислены в предложении FROM. Когда в качестве возвращаемого столбца указывается имя столбца из таблицы БД, то СУБД просто берет значение

этого столбца для каждой из строк таблицы БД и помещает его в соответствующую строку таблицы результатов запроса.

Константа, записанная в списке возвращаемых столбцов, показывает, что в каждой строке таблицы результатов запроса должно содержаться одно и то же значение.

Выражение показывает, что СУБД должна вычислять значение, помещаемое в таблицу результатов запроса, по формуле, заданной выражением.

Например, для учебной БД, показанной на рис. 2.11, в результате выполнения оператора

```
SELECT 'ГРАЖДАНИН', FIO, 'ПОЛУЧАЕТ', SUMD,
'ПЛАТИТ', SUMD*0.13 FROM PERSON
```

будет создана таблица, показанная в табл. 3.2.

Таблица 3.2

Денежные доходы граждан

ГРАЖДАНИН	FIO	ПОЛУЧАЕТ	SUMD	ПЛАТИТ	SUMD*0.13
ГРАЖДАНИН	ИВАНОВ	ПОЛУЧАЕТ	1000.00р.	ПЛАТИТ	130.00р.
ГРАЖДАНИН	ПЕТРОВ	ПОЛУЧАЕТ	700.00р.	ПЛАТИТ	91.00р.
…	…	…	…	…	…
ГРАЖДАНИН	САХАРОВ	ПОЛУЧАЕТ	NULL	ПЛАТИТ	NULL
…	…	…	…	…	…

Количество строк в таблице результатов запроса может быть любым, в частности, равным нулю. Например, выполнение оператора

```
SELECT FIO FROM PERSON WHERE NOM<0
```

приведет к созданию таблицы, в которой будет один столбец (FIO) и ни одной строки, поскольку жители не имеют отрицательных номеров.

В некоторых случаях результатом запроса может быть единственное значение, например:

```
SELECT AVG (SUMD) FROM PERSON
```

Этот оператор вычисляет среднее значение дохода, получаемого жителями, и такой результат также считается таблицей, которая состоит из одного столбца и одной строки.

То, что SQL-запрос всегда возвращает таблицу данных, очень важно с практической точки зрения. Во-первых, результаты за-

проса можно записать обратно в БД в виде таблицы. Во-вторых, таблицы результатов двух запросов, имеющие похожую структуру, можно объединить в одну таблицу. И в-третьих, таблица результатов запроса сама может стать объектом дальнейших запросов. Таким образом, табличная структура реляционной БД тесно связана с реляционными запросами: к таблицам можно посылать запросы, а запросы возвращают таблицы.

Нередко требуется получить содержимое всех столбцов таблицы, чтобы ознакомиться с ее структурой и хранимыми в ней данными. С учетом этого в операторе SELECT разрешается использовать символ звездочки, который означает, что требуется прочитать все столбцы из таблиц, перечисленных в предложении FROM. Например, содержимое таблицы PERSON можно получить с помощью оператора

```
SELECT * FROM PERSON
```

В этом случае таблица результатов запроса будет содержать все 6 столбцов таблицы PERSON, которые расположены в том же порядке, что и в исходной таблице, хранящейся в БД.

По умолчанию в таблицу результатов запроса включаются все строки, в том числе и повторяющиеся (см. ключевое слово ALL на синтаксической диаграмме рис. 3.4). Например, оператор

```
SELECT ADR FROM PERSON
```

возвращает адреса всех жителей, и среди этих адресов будут повторяющиеся, если в одной квартире проживают несколько человек.

Чтобы в таблице результатов запроса содержались только неповторяющиеся строки, в операторе SELECT перед списком возвращаемых столбцов записывается ключевое слово DISTINCT. Например, чтобы получить список разных адресов квартир, предыдущий оператор нужно дополнить этим ключевым словом:

```
SELECT DISTINCT ADR FROM PERSON
```

3.4.4. Отбор строк из таблиц

SQL-запросы, считывающие из таблицы все строки, полезны при просмотре базы данных и создании итоговых отчетов.

Однако чаще требуется выбрать из таблицы базы данных несколько строк и включить в таблицу результатов запроса только их. Чтобы указать, какие строки требуется отобрать, следует использовать предложение WHERE.

Предложение WHERE содержит условие поиска, определяющее, какие именно строки требуется прочитать. Например, чтобы получить информацию о том, кто проживает в квартире по адресу «802-12», нужно выполнить оператор

```
SELECT FIO FROM PERSON WHERE ADR='802 - 12'
```

Фактически условие поиска служит фильтром для строк таблицы. Строки, удовлетворяющие условию поиска, проходят через фильтр и становятся частью таблицы результатов запроса (рис. 3.5).

В SQL используется множество условий поиска, позволяющих эффективно и естественным образом создавать различные типы запросов. В частности, предусмотрены перечисленные далее условия поиска, называемые в стандарте SQL1 предикатами.

1. **Сравнение.** Значение одного выражения сравнивается со значением другого выражения.

2. **Проверка на принадлежность диапазону значений.** Проверяется, попадает ли указанное значение в определенный диапазон значений.

3. **Проверка на принадлежность множеству.** Проверяется, совпадает ли значение выражения с одним из значений, имеющихся в заданном множестве.

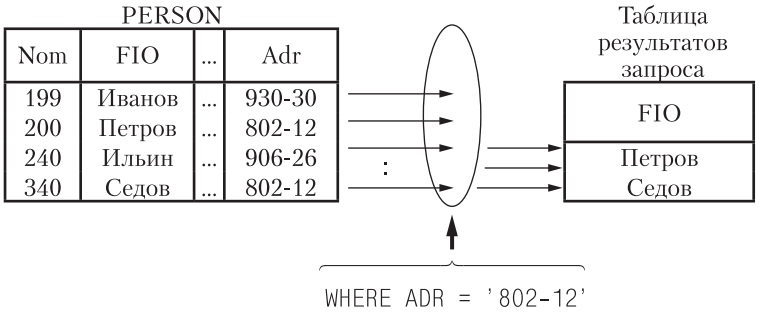


Рис. 3.5. **Предложение WHERE в роли фильтра**

4. **Проверка на соответствие шаблону.** Проверяется, соответствует ли строковое значение, содержащееся в столбце, определенному шаблону.

5. **Проверка на равенство значению NULL.** Проверяется, содержится ли в столбце значение NULL.

Сравнение

Имеется шесть различных способов сравнения значений двух выражений (рис. 3.6). Например, можно вывести список жителей, родившихся после определенной даты:

```
SELECT FIO, ADR FROM PERSON
WHERE RDATE >'12/31/1970'
```

СУБД при сравнении значений двух выражений может вычислить один из таких результатов:

1) если сравнение истинно, то результат проверки имеет значение TRUE;

2) если сравнение ложно, то результат проверки имеет значение FALSE;

3) если хотя бы одно из двух выражений имеет значение NULL, то результатом проверки будет значение NULL.

В таблицу результатов запроса включаются только те строки, для которых условие поиска имеет значение TRUE. Например, если в предложении WHERE можно было бы задать условие SUMD = NULL, то в таблице результатов не оказалось бы ни одной строки, поскольку в этом случае результатом проверки был бы NULL для любых значений, имеющихся в столбце SUMD. Поэтому, чтобы не искушать пользователя, в сравнениях запрещено использовать ключевое слово NULL.

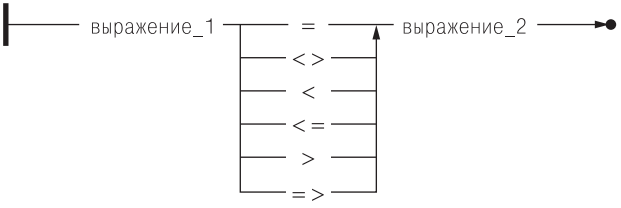


Рис. 3.6. **Синтаксическая диаграмма сравнения**

Проверка на принадлежность диапазону значений

В условие поиска этого вида входят три выражения (рис. 3.7):

1) выражение, которое задает проверяемое значение;
2) выражение, задающее нижний предел проверяемого диапазона;
3) выражение, задающее верхний предел проверяемого диапазона.

Типы данных во всех трех выражениях должны быть сравнимыми.

Рис. 3.7. **Синтаксическая диаграмма проверки на принадлежность диапазону значений**

Например, формирование списка жителей, родившихся в I квартале 1955 г., задается оператором

```
SELECT FIO, ADR FROM PERSON
WHERE RDATE BETWEEN '01/01/1955' AND '03/31/1955'
```

При проверке на принадлежность диапазону верхний и нижний пределы считаются частью диапазона.

Для проверки выхода значения за пределы диапазона следует использовать ключевое слово NOT. Например, список жителей, которые будут платить налоги меньше 100 руб. или больше 1200 руб., можно получить с помощью оператора

```
SELECT FIO, ADR FROM PERSON
WHERE (SUMD*0.13) NOT BETWEEN 100.00 AND 1200.00
```

В стандарте SQL1 определены следующие правила обработки значений NULL при проверке принадлежности диапазону (рис. 3.8):

1) если проверяемое выражение (*а*) имеет значение NULL либо оба выражения (*б*), определяющие диапазон, имеют значения NULL, то проверка BETWEEN возвращает значение NULL;

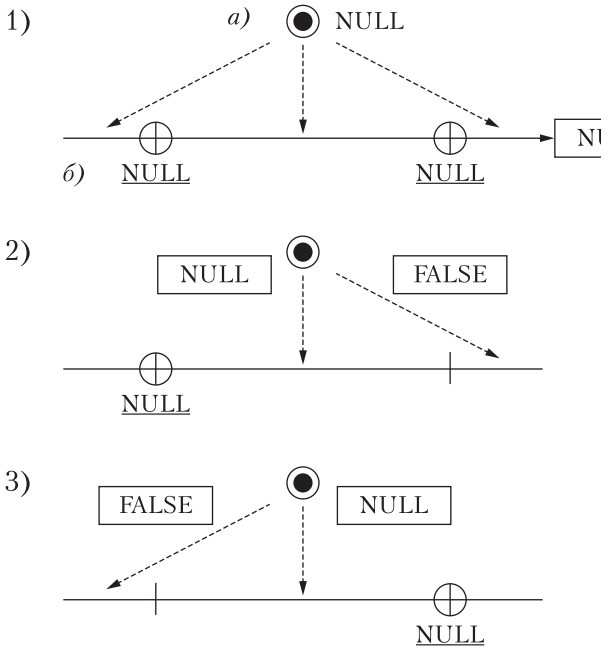


Рис. 3.8. **Правила обработки значений NULL**

2) если выражение, определяющее нижний предел диапазона, имеет значение NULL, то проверка BETWEEN возвращает значение FALSE тогда, когда проверяемое значение больше верхнего предела диапазона, и значение NULL в противном случае;

3) если выражение, определяющее верхний предел диапазона, имеет значение NULL, то проверка BETWEEN возвращает значение FALSE, если проверяемое значение меньше нижнего предела, и значение NULL в противном случае.

Проверка на принадлежность множеству

С помощью этого условия поиска можно узнать, соответствует ли значение данных какому-либо значению из заданного списка (рис. 3.9).

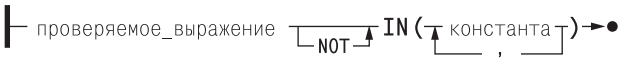


Рис. 3.9. **Синтаксическая диаграмма проверки на принадлежность множеству**

Например, следующий оператор выводит список жителей с заданными номерами:

```
SELECT * FROM PERSON
WHERE NOM IN (1,3,5,9)
```

Список жителей, родившихся в указанные дни, формируется оператором

```
SELECT * FROM PERSON
WHERE RDATE IN ('02/12/55','11/11/11')
```

С помощью проверки NOT IN можно обнаружить значения данных, не являющиеся членами заданного множества (см. учебную БД на рис. 2.11):

```
SELECT * FROM FLAT
WHERE KCATEGORY NOT IN ('П','К')
```

В результате будут выбраны строки таблицы FLAT со сведениями о квартирах категории 'Н', т.е. о неприватизированных квартирах.

Проверяемое выражение может быть любым допустимым выражением, однако обычно оно представляет собой имя столбца.

Если результатом проверяемого выражения является значение NULL, то проверка возвращает значение NULL.

Все элементы в списке проверяемых значений должны быть одного и того же типа данных, который должен быть сравним с типом данных проверяемого выражения.

Проверка на соответствие шаблону

Для формирования таблицы результатов, состоящей из строк, в которых значение некоторого текстового столбца (поля) совпадает с заданным текстом, можно использовать простое сравнение. Например, следующий оператор создает таблицу результатов из строк, в которых встречаются заданные фамилия имя и отчество жителя:

```
SELECT * FROM PERSON
WHERE FIO = 'Иванов Иван Иванович'
```

Однако, если необходимо получить сведения о всех Ивановых, то следует воспользоваться проверкой на соответствие шаблону (рис. 3.10).

Рис. 3.10. **Синтаксическая диаграмма проверки на соответствие шаблону**

Эта конструкция позволяет определить, соответствует ли значение данных в столбце некоторому шаблону. Шаблон представляет собой строку символов, в которую может входить один или более подстановочных знаков. Эти знаки интерпретируются особым образом.

Подстановочный знак «%» совпадает с любой последовательностью из нуля или более символов. Например, чтобы получить сведения обо всех Ивановых, нужно воспользоваться шаблоном:

```
SELECT * FROM PERSON
WHERE FIO LIKE 'Иванов%'
```

Ключевое слово LIKE указывает, что необходимо сравнивать содержимое столбца FIO с шаблоном 'Иванов%', которому соответствуют все фамилии, начинающиеся словом «Иванов», в том числе «Ивановский». Чтобы получить сведения только о мужчинах Ивановых или только о женщинах Ивановых, следует использовать шаблон 'Иванов %' или 'Иванова %' соответственно.

Подстановочный знак «_» (подчеркивание) совпадает с любым отдельным символом. Например, для получения сведений о жителях, проживающих в 1-м микрорайоне, можно воспользоваться таким шаблоном:

```
SELECT * FROM PERSON
WHERE ADR LIKE 'Зеленоград, 1_ _-%'
```

Подстановочные знаки можно помещать в любое место шаблона, и в одном шаблоне может содержаться несколько подстановочных знаков.

С помощью конструкции NOT LIKE можно выбирать строки, которые не соответствуют шаблону.

Проверка на соответствие шаблону применима только к столбцам, имеющим строковый тип данных или содержащим дату и время. Если в столбце содержится значение NULL, то результатом проверки будет значение NULL.

Может случиться так, что среди символов, содержащихся в столбце, окажутся в качестве отдельных символов подстановочные знаки:

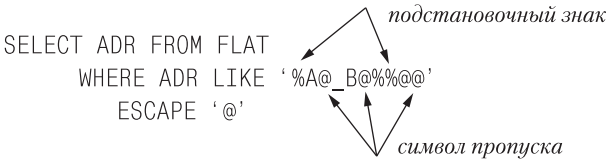
X	*Y*	*Z*
.....	AB%CD	.....
.....	A_D%E	.....

В стандарте SQL1 определен способ проверки наличия в строке литер, используемых в качестве подстановочных знаков. Для этого применяются символы пропуска.

Когда в шаблоне встречается символ пропуска, то символ, следующий непосредственно за ним, считается не подстановочным знаком, а обычной литерой, т.е. происходит пропуск символа.

Непосредственно за символом пропуска может следовать либо один из двух подстановочных знаков, либо сам символ пропуска, поскольку он тоже приобретает в шаблоне особое значение.

Символ пропуска определяется в предложении ESCAPE в виде строки, состоящей из одного символа. В следующем примере в качестве символа пропуска используется литера «@», и шаблон обеспечивает выборку адресов, в которых встречается сочетание символов «A_B%» и которые заканчиваются литерой «@»:



Возможность использования символов пропуска реализована не во всех СУБД, поэтому применения предложения ESCAPE стараются избегать.

Проверка на равенство значению NULL

Результатом вычисления рассмотренных условий поиска может одно из трех значений: TRUE, FALSE или NULL (например, если один из столбцов содержит значение NULL). В ряде случаев бывает необходимо проверять значения в столбцах на равенство значению NULL и помещать соответствующие строки в таблицу результатов. Для этого в языке SQL предусмотрена специальная проверка на равенство значению NULL (рис. 3.11).

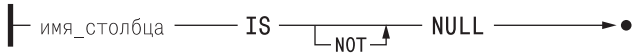


Рис. 3.11. **Синтаксическая диаграмма проверки на равенство значению NULL**

В следующем операторе проверка на NULL используется для определения свободных номеров телефонов:

```
SELECT * FROM TPHONE
WHERE ADR IS NULL
```

Инвертированная форма проверки на NULL позволяет отыскать строки, которые не содержат значения NULL. Например, список номеров телефонов, установленных в квартирах, формирует оператор

```
SELECT * FROM TPHONE
      WHERE ADR IS NOT NULL
```

В отличие от условий поиска, рассмотренных ранее, проверка на NULL не может возвратить значение NULL в качестве результата. Она всегда возвращает TRUE или FALSE.

Составные условия поиска

Все описанные простые условия поиска можно объединить в более сложные с помощью ключевых слов AND, OR, NOT. Возможность использования составных условий поиска предусмотрена правилами записи предложения WHERE (рис. 3.12), уточняющими ранее приведенную на рис. 3.4 форму этого предложения в составе оператора SELECT.

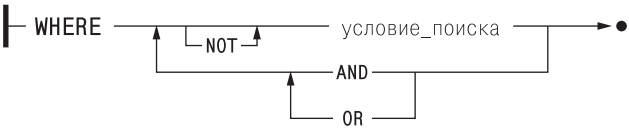


Рис. 3.12. **Синтаксическая диаграмма предложения WHERE**

Ключевые слова AND, OR, NOT являются операторами логических операций И, ИЛИ, НЕ. С помощью этих операторов и круглых скобок можно создавать очень сложные условия поиска. Например, проверку на принадлежность множеству NOM IN (1, 3, 5, 9) можно заменить другим (менее удачным) условием поиска:

(NOM = 1) OR (NOM = 3) OR (NOM = 5) OR (NOM = 9)

Таблицы истинности для операторов AND, OR, NOT учитывают наличие значения NULL (табл. 3.3).

Таблица 3.3

Таблицы истинности

Таблица AND	**TRUE**	**FALSE**	**NULL**
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL
Таблица OR	**TRUE**	**FALSE**	**NULL**
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL
Таблица NOT	**TRUE**	**FALSE**	**NULL**
	FALSE	TRUE	NULL

3.4.5. Сортировка таблицы результатов запроса

Строки таблицы результатов запроса, как и строки таблицы в БД, не имеют определенного порядка. Чтобы отсортировать таблицу результатов запроса, в оператор SELECT нужно включить предложение ORDER BY (рис. 3.13).

Например, таблица результатов следующего запроса отсортирована по двум столбцам ADR и FIO:

```
SELECT ADR, FIO, POL, SUMD FROM
PERSON ORDER BY ADR, FIO
```

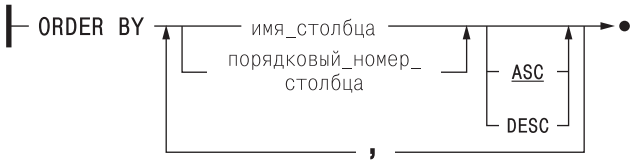


Рис. 3.13. **Синтаксическая диаграмма предложения ORDER BY**

В такой таблице легко определить жителей, проживающих в одной квартире, причем список таких жителей упорядочен по алфавиту.

Первый столбец (ADR) является главным ключом сортировки, следующие за ним (в данном примере FIO) являются все более второстепенными ключами. Можно сортировать таблицу результатов запроса по любому элементу списка возвращаемых столбцов.

В предложении ORDER BY можно задать возрастающий или убывающий порядок сортировки. По умолчанию, данные сортируются в порядке возрастания (ASC). Чтобы сортировать данные по убыванию, следует использовать ключевое слово DESC. Например:

```
SELECT * FROM PERSON ORDER BY SUMD DESC
```

Если столбец таблицы результатов, используемый для сортировки, является вычисляемым, то у него нет имени. Поэтому в таком случае вместо имени столбца необходимо указать его порядковый номер:

```
SELECT FIO, ADR, SUMD*0.13
FROM PERSON
ORDER BY 3 DESC
```

Одновременно используя имена и номера столбцов, а также возрастающий и убывающий порядки сортировки, можно сортировать таблицу результатов запроса по достаточно сложному алгоритму. Например, можно вывести список жителей, отсортированный по полу (причем сначала мужчин, затем женщин), а для каждого поля — отсортированный по возрастанию размера налога:

```
SELECT FIO, ADR, POL, SUMD*0.13
FROM PERSON
ORDER BY POL DESC, 4 ASC
```

3.4.6. Объединение результатов нескольких запросов

Иногда появляется необходимость объединения результатов двух или более запросов в одну таблицу. Язык SQL предусматривает такую возможность с помощью оператора UNION. Например, можно объединить результаты, выполнив следующий сложный оператор, и получить результат, показанный на рис. 3.14:

```
SELECT ADR, POL FROM PERSON
WHERE SUMD<2000.00
 UNION
SELECT ADR, KCATEGORY FROM FLAT
WHERE SKV<30.00
```

Чтобы таблицы результатов запроса можно было объединить с помощью оператора UNION, они должны удовлетворять следующим требованиям:

1) таблицы должны содержать одинаковое число столбцов;

2) тип данных каждого столбца первой таблицы должен совпадать с типом данных соответствующего столбца во второй таблице;

3) ни одна из таблиц не может быть отсортирована с помощью предложения ORDER BY, однако объединенные результаты запроса можно отсортировать.

Обратите внимание на то, что имена столбцов в объединяемых таблицах не обязательно должны быть одинаковыми.

Стандарт SQL1 накладывает дополнительные ограничения на операторы SELECT, участвующие в операторе UNION. Он разрешает использовать в списке возвращаемых столбцов только имена столбцов или указатель на все столбцы (SELECT *) и запрещает использовать выражения. Коммерческие реализации в большинстве случаев снимают последний запрет, но могут накладывать другие ограничения.

Поскольку оператор UNION объединяет строки из двух таблиц результатов, то вполне вероятно, что в объединенной таблице будут содержаться повторяющиеся строки. По

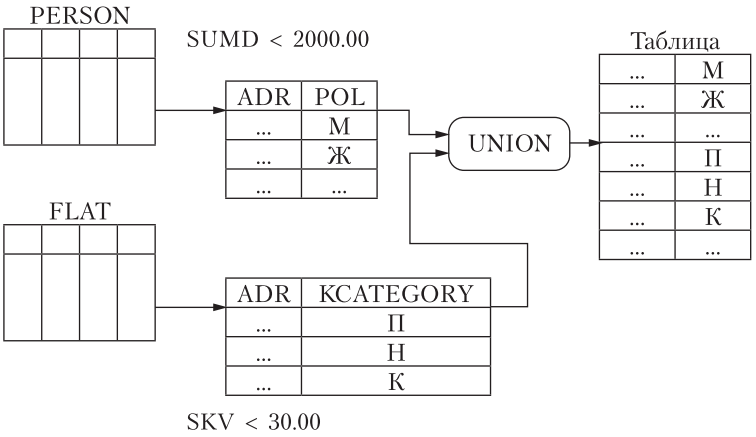


Рис. 3.14. **Использование оператора UNION**

умолчанию оператор UNION в процессе своего выполнения удаляет повторяющиеся строки.

Если в результирующей объединенной таблице необходимо сохранить повторяющиеся строки, сразу за ключевым словом UNION следует указать ключевое слово ALL.

Хотя предложение ORDER BY нельзя использовать в составляющих операторах SELECT, объединенных оператором UNION, результирующую объединенную таблицу можно отсортировать, записав предложение ORDER BY за последним оператором SELECT. Поскольку столбцы в таблице результатов запроса на объединение не имеют имен, то в предложении ORDER BY следует указывать номера столбцов.

Оператор UNION можно использовать многократно, чтобы объединить результаты трех или более запросов, например:

```
SELECT * FROM A
UNION (SELECT * FROM B UNION SELECT * FROM C)
```

Скобки показывают, какой оператор UNION должен выполняться первым.

Если используются одинаковые формы оператора UNION (т.е. только UNION либо только UNION ALL), то порядок выполнения операторов не имеет значения. Следующие операторы полностью эквивалентны:

```
A UNION (B UNION C)
(A UNION B ) UNION C
(A UNION C) UNION B
```

Эквивалентными являются и такие операторы:

```
A UNION ALL (B UNION ALL C)
(A UNION ALL B) UNION ALL C
(A UNION ALL C) UNION ALL B
```

Однако если в запросы на объединения входят как операторы UNION, так и операторы UNION ALL, то порядок следования этих операторов имеет значение. Если операторы A UNION ALL B UNION C проинтерпретировать как A UNION ALL (B UNION C), то таблица результатов может содержать повторяющиеся строки. Однако если эти операторы проинтерпретировать как (A UNION ALL B) UNION C, то в таблице результатов повторяющихся строк не будет.

По этой причине всегда нужно использовать круглые скобки, чтобы указать последовательность выполнения операторов.

3.4.7. Многотабличные запросы на чтение (соединения)

Язык SQL позволяет получать ответы на многотабличные запросы, которые соединяют данные из нескольких таблиц.

Чтобы понять, как в SQL реализуются многотабличные запросы, начнем с рассмотрения простого запроса, который соединяет данные из двух различных таблиц: вывести список жителей, имеющих телефоны, с указанием фамилии, имени, отчества, адреса квартиры и номера телефона.

Из рис. 3.15 очевидно, что три запрашиваемых элемента данных хранятся в двух различных таблицах:

1) в таблице PERSON содержатся фамилия, имя, отчество жителя и адрес его квартиры, но в ней отсутствует номер телефона;

2) в таблице TPHONE содержатся номера телефонов и адреса квартир, где они установлены, но в ней нет информации о жителях.

Однако между двумя этими таблицами существует связь по адресам квартир. Очевидно, чтобы получить требуемую таблицу результатов, в операторе SELECT надо учесть существующую связь между таблицами. Этот оператор должен выполнить предположительно следующие действия:

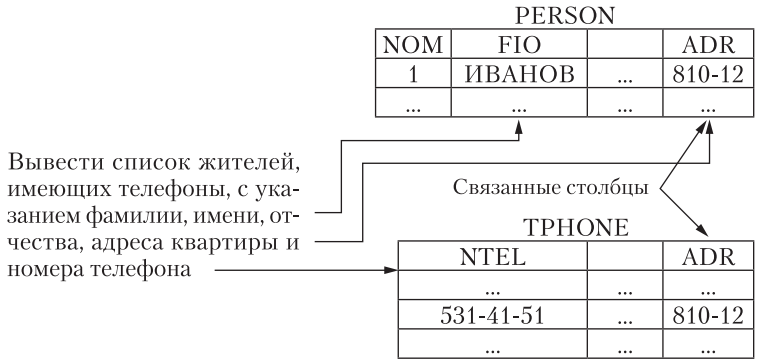


Рис. 3.15. **Запрос который соединяет данные из двух таблиц**

1) образовать пустую таблицу результатов из трех столбцов, содержащих значения FIO, ADR, NTEL, и начать просмотр таблицы TPHONE;

2) найти в таблицах TPHONE и PERSON пару строк, в которых совпадают значения столбцов ADR;

3) из найденной пары строк выбрать фамилию, имя, отчество (FIO), адрес квартиры (ADR), номер телефона (NTEL) и сформировать из этих значений одну новую строку в таблице результатов;

4) пока не просмотрены все строки в таблице TPHONE, переходить к шагу 2.

Конечно, это не единственный способ получить таблицу результатов данного запроса, но как бы она ни получалась, всегда имеются две особенности:

1) каждая строка таблицы результатов запроса формируется из пары строк: одна строка находится в таблице TPHONE, а другая — в таблице PERSON;

2) для нахождения данной пары строк производится сравнение содержимого соответствующих столбцов в этих таблицах.

Процесс формирования строк путем сравнения содержимого соответствующих столбцов называется соединением таблиц. Соединение таблиц задается в предложении WHERE, в котором условие поиска имеет вид сравнения столбцов из разных таблиц.

В нашем примере нужная таблица результатов формируется оператором:

```
SELECT FIO, ADR, NTEL
FROM PERSON, TPHONE
WHERE PERSON.ADR=TPHONE.ADR
```

Обратите внимание, что в операторе SELECT нет указаний о том, как должен выполняться запрос. В нем нет никаких упоминаний вроде «начните с жителей» или «начните с телефонов». Вместо этого в запросе указано, что должны представлять собой результаты запроса. Способ получения результатов определяет сама СУБД.

В многотабличном запросе можно комбинировать условие поиска, в котором задаются связанные столбцы, с другими условиями поиска, чтобы еще больше сузить результаты запроса.

Например, можно вывести список жителей, имеющих телефоны и получающих общий доход не меньший 1400 руб.:

```
SELECT FIO, ADR, NTEL
FROM PERSON, NTEL
WHERE PERSON.ADR=TPHONE.ADR
AND SUMD>=1400.00
```

Язык SQL позволяет соединять данные из трех или более таблиц, используя ту же методику, что и для соединения данных из двух таблиц. Например, следующий оператор выводит список жителей, имеющих телефоны и получающих общий доход не меньше 1400 руб., с указанием категорий квартиры и телефона:

```
SELECT FIO, ADR, KCATEGORY, NTEL, TCATEGORY
FROM PERSON, FLAT, TPHONE
WHERE PERSON.ADR=FLAT.ADR
AND FLAT.ADR=TPHONE.ADR
AND SUMD>=1400.00
```

Условия поиска, задающие связь столбцов, отражают соответствие между первичными и внешними ключами таблиц (см. рис. 2.11).

В БД имеется таблица PROFIT, хранящая перечень различных источников (SOURCE) и размеров (MONEY) доходов, пронумерованных с использованием числового идентификатора ID, и таблица HAVE_D, показывающая связь жителя (NOM) с его доходами (ID), поэтому можно получить ответ на

запрос обо всех доходах, которые получает житель с заданным номером (или известной фамилией, именем, отчеством):

```
SELECT NOM, FIO, SOURCE, MONEY
FROM PERSON, PROFIT, HAVE_D
WHERE PERSON.NOM=HAVE_D.NOM
AND PROFIT.ID=HAVE_D.ID
AND PERSON.NOM=12
```

В многотабличных запросах, чтобы исключить неоднозначные ссылки на столбцы, может потребоваться указание полных имен в списке возвращаемых столбцов оператора SELECT. Чтобы не набирать длинные имена таблиц перед именами столбцов, в предложении FROM для таблиц можно задать *псевдонимы*, а затем использовать псевдоним вместо имени таблицы в полном имени столбца:

```
SELECT A.NOM, A.FIO, B.SOURCE, B.MONEY
FROM PERSON A, PROFIT B, HAVE_D C
WHERE A.NOM=C.NOM AND B.ID=C.ID
AND A.NOM=12
```

Псевдонимом может быть любое допустимое имя.

В многотабличном запросе вместо списка возвращаемых столбцов может быть использована звездочка «*», которая означает включение в таблицу результатов всех столбцов из всех таблиц, указанных в предложении FROM. Например, таблица результатов следующего запроса состоит из девяти столбцов (шесть — из PERSON, три — из TPHONE):

```
SELECT * FROM PERSON, TPHONE
WHERE PERSON.ADR=TPHONE.ADR
```

Многие реализации языка SQL трактуют звездочку как особый вид универсального имени столбца, которое распространяется на все столбцы. В этих реализациях звездочка вместе с именем используется вместо списка полных имен столбцов. В следующем запросе таблица результатов содержит все столбцы PERSON и один столбец из таблицы TPHONE:

```
SELECT PERSON.*, NTEL
FROM PERSON, TPHONE
WHERE PERSON.ADR=TPHONE.ADR
```

Самосоединение

Возможность использования псевдонимов для таблиц позволяет соединить таблицу саму с собой. Например, найти пары жителей-однофамильцев, не проживающих вместе, позволяет следующий запрос:

```
SELECT A.FIO, A.ADR, COPY.FIO, COPY.ADR
FROM PERSON A, PERSON COPY
WHERE A.FIO=COPY.FIO
AND A.ADR<>COPY.ADR
```

Другой пример запроса позволяет определить все пары жителей, получающих одинаковый по размеру общий доход:

```
SELECT A.FIO, COPY.FIO, A.SUMD
FROM PERSON A, PERSON COPY
WHERE A.SUMD=COPY.SUMD
```

Из-за симметричности условия поиска таблица результатов содержит избыточные данные для каждой пары жителей *X* и *Y*:

PERSON.FIO	COPY.FIO	PERSON.SUMD
X	*X*	*N*
X	*Y*	*N*
Y	*X*	*N*
Y	*Y*	*N*

Из каждой четверки строк интерес представляет только одна строка, выделенная пунктиром.

Чтобы исключить избыточность, условие поиска нужно дополнить проверкой PERSON.FIO < COPY.FIO, чтобы сделать условие поиска асимметричным:

```
SELECT A.FIO, COPY.FIO, A.SUMD
FROM PERSON A, PERSON COPY
```

```
WHERE A.SUMD=COPY.SUMD
AND A.FIO<COPY.FIO
```

С увеличением количества таблиц в запросе резко возрастает объем работы, необходимой для выполнения запроса. В самом языке SQL нет ограничений на число таблиц, указанных в одном запросе. Но некоторые реализации SQL ограничивают число таблиц; чаще всего их количество ограничивается восемью.

3.4.8. Итоговые запросы на чтение

Многие запросы к БД не нуждаются в той степени детализации, которую обеспечивают SQL-запросы, рассмотренные в предыдущих примерах. Так, во всех запросах перечисленных далее, требуется узнать всего одно или несколько значений, которые подытоживают информацию, содержащуюся в БД:

1) какова сумма доходов у всех жителей?
2) каков наибольший и наименьший общий доход отдельного жителя?
3) каков среднедушевой доход жителя Зеленограда?
4) каков среднедушевой доход жителей каждой квартиры?
5) сколько жителей в каждой квартире?

На языке SQL запросы такого типа можно создавать с помощью агрегатных функций и предложений GROUP BY и HAVING, используемых в операторе SELECT.

Использование агрегатных функций

Для подведения итогов по информации, содержащейся в БД, в SQL предусмотрены агрегатные функции. Агрегатная функция принимает в качестве аргумента какой-либо столбец данных целиком, а возвращает одно значение, которое определенным образом подытоживает этот столбец.

Например, агрегатная функция AVG() принимает в качестве аргумента столбец чисел и вычисляет их среднее значение.

Чтобы вычислить среднедушевой доход жителя Зеленограда, нужен такой запрос:

```
SELECT 'СРЕДНЕДУШЕВОЙ ДОХОД=', AVG(SUMD)
FROM PERSON
```

В SQL имеется шесть агрегатных функций, которые позволяют получать различные виды итоговой информации (рис. 3.16):

— SUM( ) вычисляет сумму всех значений, содержащихся в столбце;

— AVG( ) вычисляет среднее среди значений, содержащихся в столбце;

— MIN( ) находит наименьшее среди всех значений, содержащихся в столбце;

— MAX( ) находит наибольшее среди всех значений, содержащихся в столбце;

— COUNT( ) подсчитывает количество значений, содержащихся в столбце;

— COUNT(*) подсчитывает количество строк в таблице результатов запроса.

Аргументом агрегатной функции может быть простое имя столбца, как в предыдущем примере, или выражение, как в следующем запросе, задающем вычисление среднедушевого налога:

```
SELECT AVG(SUMD*0.13)
FROM PERSON
```

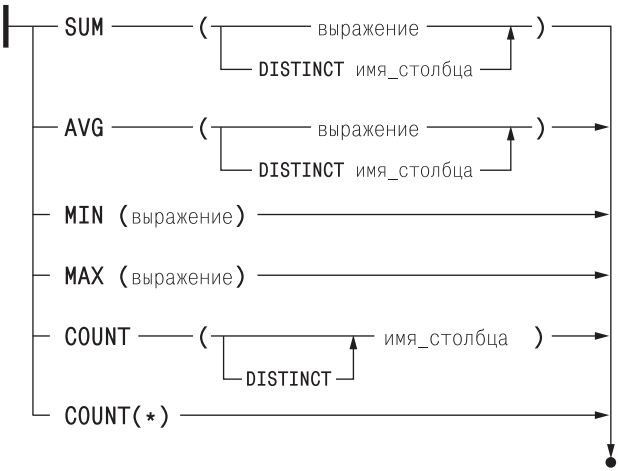


Рис. 3.16. **Синтаксическая диаграмма агрегатных функций**

При выполнении этого запроса создается временный столбец, содержащий значения (SUMD*0.13) для каждой строки таблицы PERSON, а затем вычисляется среднее значение временного столбца.

Сумму доходов у всех жителей Зеленограда можно вычислить с помощью агрегатной функции SUM:

```
SELECT SUM(SUMD) FROM PERSON
```

Агрегатная функция может быть использована и для вычисления итогов по таблице результатов, полученной соединением нескольких исходных таблиц. Например, можно вычислить общую сумму дохода, которая получена жителями от источника с названием «Стипендия»:

```
SELECT SUM(MONEY)
FROM PROFIT, HAVE_D
WHERE PROFIT.ID=HAVE_D.ID
AND PROFIT.SOURCE='Стипендия'
```

Агрегатные функции MIN( ) и MAX( ) позволяют найти соответственно наименьшее и наибольшее значения в таблице. При этом столбец может содержать числовые или строковые значения либо значения даты или времени.

Например, можно определить:

(а) наименьший общий доход, полученный жителями, и наибольший налог, подлежащий уплате:

```
SELECT MIN(SUMD), MAX(SUMD*0.13)
FROM PERSON
```

(б) даты рождения самого старого и самого молодого жителя:

```
SELECT MIN(RDATE), MAX(RDATE)
FROM PERSON
```

(в) фамилии, имена и отчества самого первого и самого последнего жителей в списке, упорядоченном по алфавиту:

```
SELECT MIN(FIO), MAX(FIO)
FROM PERSON
```

Применяя эти агрегатные функции, нужно помнить, что числовые данные сравниваются по арифметическим правилам, сравнение дат происходит последовательно (более ранние значения дат считаются меньшими, чем более поздние), сравнение интервалов времени выполняется на основании их продолжительности.

При использовании функции MIN( ) и MAX( ) со строковыми данными результат сравнения двух строк зависит от используемой таблицы кодировки символов.

Агрегатная функция COUNT( ) подсчитывает количество значений в столбце любого типа:

(а) сколько квартир в 1-м микрорайоне?

```
SELECT COUNT(ADR)
FROM FLAT
WHERE ADR LIKE '%, 1_ _-%'
```

(б) сколько жителей имеют источники дохода?

```
SELECT COUNT(DISTINCT NOM)
FROM HAVE_D
```

(в) сколько источников дохода используются жителями?

```
SELECT COUNT(DISTINCT ID)
FROM HAVE_D
```

Ключевое слово «DISTINCT» указывает, что подсчитываются неповторяющиеся значения в столбце.

Специальная агрегатная функция COUNT(*) подсчитывает строки в таблице результатов, а не значения данных:

(а) сколько квартир во 2-м микрорайоне?

```
SELECT COUNT(*)
FROM FLAT
WHERE ADR LIKE '%, 2__-%'
```

(б) сколько источников дохода у Иванова Ивана Ивановича?

```
SELECT COUNT(*)
FROM PERSON, HAVE_D
```

```
WHERE FIO = 'Иванов Иван Иванович'
AND PERSON.NOM = HAVE_D.NOM
```

(в) сколько жителей проживает в квартире по определенному адресу?

```
SELECT COUNT(*) FROM PERSON
WHERE ADR = 'Зеленоград, 1001-45'
```

Один из способов понять, как выполняются итоговые запросы с агрегатными функциями, — это представить выполнение запроса разбитым на две части. Сначала определяется, как бы запрос работал без агрегатных функций, возвращая несколько строк результатов. Затем применяются агрегатные функции к результатам запроса, возвращая одну итоговую строку.

Например, рассмотрим следующий сложный запрос: найти среднедушевой общий доход, сумму общих доходов жителей, а также среднюю доходность источника в процентах от общего дохода жителя. Ответ дает оператор

```
SELECT AVG(SUMD), SUM(SUMD), (100*AVG(MONEY/SUMD))
FROM PERSON, PROFIT, HAVE_D
WHERE PERSON.NOM = HAVE_D.NOM
AND HAVE_D.ID = PROFIT.ID
```

Без агрегатных функций запрос выглядел бы так:

```
SELECT SUMD, SUMD, MONEY/SUMD
FROM PERSON, PROFIT, HAVE_D
WHERE PERSON.NOM = HAVE_D.NOM
AND HAVE_D.ID = PROFIT.ID
```

и возвращал бы одну строку результатов для каждого жителя и конкретного источника дохода. Агрегатные функции используют столбцы таблицы результатов этого запроса для получения однострочной таблицы с итоговыми результатами.

В строке возвращаемых столбцов вместо имени любого столбца можно указать агрегатную функцию. Например, она может входить в выражение, в котором суммируются или вычитаются значения двух агрегатных функций:

```
SELECT MAX(SUMD)-MIN(SUMD)
FROM PERSON
```

Однако агрегатная функция не может быть аргументом для другой агрегатной функции, т.е. запрещены вложенные агрегатные функции.

Кроме того, в списке возвращаемых столбцов нельзя одновременно использовать агрегатные функции и обычные имена столбцов, поскольку в этом нет смысла, например:

```
SELECT FIO, SUM(SUMD)
FROM PERSON
```

Здесь первый элемент списка указывает, чтобы СУБД создала таблицу, которая будет состоять из нескольких строк и содержать по одной строке для каждого жителя. Второй элемент списка просит СУБД получить одно результирующее значение, являющееся суммой значений столбца SUMD. Эти два указания противоречат друг другу, что приводит к ошибке.

По этой причине либо все ссылки на столбцы в списке возвращаемых столбцов должны являться аргументами агрегатных функций, либо в списке не должно быть ни одной агрегатной функции.

Сказанное не относится к случаям обработки подзапросов и запросов с группировкой.

Агрегатные функции и значения NULL

Агрегатные функции (кроме COUNT(*)) в качестве аргумента принимают столбец значений и возвращают в качестве результата одно значение. Если в столбце значений встречаются значения NULL, то они игнорируются агрегатными функциями.

В стандарте SQL1 определены следующие правила обработки значений NULL в агрегатных функциях:

1) если какие-либо из значений, содержащихся в столбце, равны NULL, то при вычислении результата функции они исключаются;

2) если все значения в столбце равны NULL или в столбце нет значений (т.е. столбец пустой), то функции SUM( ), AVG( ), MIN( ), MAX( ) возвращают значение NULL; функция COUNT( ) возвращает нулевое значение;

3) функция COUNT(*) подсчитывает количество строк и не зависит от наличия или отсутствия в столбце значений NULL; если строк в таблице нет, то функция возвращает нулевое значение.

Прежде чем полагаться на эти правила, следует проверить свою СУБД, поскольку коммерческие СУБД могут выдавать результаты, отличающиеся от стандарта.

Удаление повторяющихся строк (DISTINCT)

С помощью ключевого слова DISTINCT, записанного перед аргументом агрегатной функции, можно указать, что перед применением агрегатной функции к столбцу из него следует удалить все повторяющиеся значения.

3.4.9. Запросы с группировкой

Рассмотренные примеры итоговых запросов относятся ко всей таблице в целом и в качестве результата возвращают одну строку. Например, запрос

```
SELECT AVG(SUMD) FROM PERSON
```

вычисляет среднедушевой доход жителя.

Наряду с такими «интегральными» итоговыми запросами большой интерес представляют итоговые запросы, в которых агрегатные функции применяются к определенным группам строк, а не ко всей таблице. Эту возможность предоставляет предложение GROUP BY оператора SELECT. Например, определить среднедушевой доход жителей каждой квартиры позволяет такой запрос:

```
SELECT ADR,AVG(SUMD) FROM PERSON
GROUP BY ADR
```

Этот запрос возвращает несколько итоговых строк — по одной строке для каждой квартиры. Логика выполнения запроса следующая.

1. Сведения о жителях в таблице PERSON делятся на группы, по одной группе на каждую квартиру. В каждой группе все жители имеют одно и то же значение в столбце ADR.
2. Для каждой группы вычисляется среднее значение столбца SUMD по всем строкам, входящим в группу, и ге-

нерируется одна итоговая строка в таблице результатов. Эта строка содержит значение столбца ADR для группы и средний общий доход для данной группы.

Запрос, включающий в себя предложение GROUP BY, называется *запросом с группировкой*. Столбцы, указанные в этом предложении, называются *столбцами группировки*, поскольку именно они определяют, по какому признаку строки объединяются в группы. Ниже приведен ряд запросов с группировкой:

1) сколько жителей зарегистрировано в каждой квартире?

```
SELECT ADR,COUNT(*)
FROM PERSON
GROUP BY ADR
```

2) сколько источников дохода имеет каждый житель с ненулевым общим доходом?

```
SELECT NOM,COUNT(ID)
FROM HAVE_D
GROUP BY NOM
```

3) сколько различных источников дохода имеют жители каждой квартиры?

```
SELECT ADR,COUNT(DISTINCT ID)
FROM PERSON,HAVE_D
WHERE PERSON.NOM = HAVE_D.NOM
GROUP BY ADR
```

Предложение GROUP BY видоизменяет действие агрегатных функций. Это предложение указывает, что результаты запроса следует разделить на группы, применить агрегатную функцию по отдельности к каждой группе и получить для каждой группы одну строку в таблице результатов.

SQL позволяет группировать строки с учетом двух или более столбцов. Например, можно сгруппировать жителей по адресам и фамилиям и подсчитать сумму общих доходов у однофамильцев и полных тезок, проживающих вместе:

```
SELECT ADR,FIO,SUM(SUMD)
FROM PERSON
GROUP BY ADR,FIO
```

Ограничения на запросы с группировкой

Столбцы группировки, указанные в предложении GROUP BY, должны быть разными столбцами таблиц, перечисленных в предложении FROM. Нельзя группировать строки на основании значения вычисляемого выражения.

Следует учитывать также ограничения на элементы списка возвращаемых столбцов. Все элементы этого списка должны иметь одно значение для каждой группы строк. Это означает, что возвращаемым столбцом может быть:

1) константа;

2) агрегатная функция, возвращающая одно значение для всех строк, входящих в группу;

3) столбец группировки, который, по определению, имеет одно и то же значение во всех строках группы;

4) выражение, включающее в себя перечисленные выше элементы.

Например, следующий запрос для определения количества источников дохода у каждого жителя с ненулевым общим доходом не соответствует приведенным ограничениям:

```
SELECT NOM,FIO,SUMD,COUNT(ID)
FROM PERSON,HAVE_D
WHERE PERSON.NOM = HAVE_D.NOM
GROUP BY NOM
```

В этом запросе список возвращаемых столбцов содержит помимо столбца группировки NOM также столбцы FIO и SUMD, не указанные в предложении GROUP BY. Чтобы не нарушать ограничение, необходимо просто включить эти два столбца в предложение GROUP BY:

```
SELECT NOM,FIO,SUMD,COUNT(ID)
FROM PERSON,HAVE_D
WHERE PERSON.NOM = HAVE_D.NOM

GROUP BY NOM,FIO,SUMD
```

Условия поиска групп

Точно так же, как предложение WHERE используется для отбора отдельных строк, участвующих в запросе, предложение HAVING можно применить для отбора групп строк. Например, ранее рассмотренный запрос для определения среднедушевого дохода жителей каждой квартиры можно модифицировать с помощью предложения HAVING, чтобы получить информацию только о квартирах, в которых проживает более одного человека:

```
SELECT ADR,AVG(SUMD) FROM PERSON
GROUP BY ADR
HAVING COUNT(*)>1
```

Запрос выполняется в такой последовательности. Вначале предложение GROUP BY разделяет жителей на группы по адресам. После этого предложение HAVING исключает все группы, в которых количество строк равно единице. И наконец, предложение SELECT вычисляет среднедушевой доход для каждой из оставшихся групп и генерирует таблицу результатов запроса.

Ограничения на условия поиска групп

Условие поиска, используемое в предложении HAVING, применяется не к отдельным строкам, а к группе в целом. Это значит, что в условие поиска может входить:

1) константа;

2) агрегатная функция, возвращающая одно значение для всех строк, входящих в группу;

3) столбец группировки, который, по определению, имеет одно и то же значение во всех строках группы;

4) выражение, включающее в себя перечисленные выше элементы.

На практике условие поиска в предложении HAVING всегда должно включать в себя как минимум одну агрегатную функцию. Если это не так, то условие поиска можно переместить в предложение WHERE. Чтобы определить, где следует указать условие поиска, необходимо помнить, как применяются эти предложения:

1) предложение WHERE применяется к отдельным строкам, поэтому выражения, содержащиеся в нем, должны вычисляться для отдельных строк;

2) предложение HAVING применяется к группам строк, поэтому выражения, содержащиеся в нем, должны вычисляться для групп строк.

3.4.10. Вложенные запросы на чтение

В языке SQL существует понятие вложенного запроса. Механизм вложенных запросов позволяет использовать результат одного запроса в качестве составной части другого запроса.

Возможность применения одного запроса внутри другого и была причиной появления слова «структурированный» в названии языка SQL. Понятие вложенного запроса играет важную роль в SQL по трем причинам:

1) оператор языка SQL с вложенным запросом зачастую является самым естественным способом выражения запроса, так как он лучше всего соответствует словесному описанию запроса;

2) вложенные запросы облегчают написание операторов SELECT, поскольку они позволяют разбивать запросы на части (т.е. на запрос и вложенные запросы), а затем складывать эти части вместе;

3) существуют запросы, которые нельзя сформулировать, не прибегая к помощи вложенных запросов.

Вложенным, или подчиненным, запросом, называется запрос, содержащийся в предложении WHERE или HAVING другого оператора. Вложенные запросы позволяют естественным образом формулировать запросы, которые используют результаты других запросов.

Рассмотрим следующий запрос: вывести список жителей, у которых общий доход общий доход меньше размера самого доходного источника. Если известен размер *S* самого доходного источника, то запрос выражается достаточно просто:

```
SELECT NOM,FIO FROM PERSON
WHERE SUMD<S
```

К сожалению, в таком виде оператор SELECT не может быть выполнен из-за некорректного условия поиска, содер-

жащего величину *S*. Однако эту величину можно легко вычислить таким оператором:

```
SELECT MAX(MONEY) FROM PROFIT
```

Этот оператор следует использовать вместо величины *S* в качестве вложенного запроса:

```
SELECT NOM, FIO FROM PERSON
WHERE SUMD < (SELECT MAX(MONEY) FROM PROFIT)
```

Это и есть правильный SQL-запрос. Вложенный (внутренний) запрос вычисляет размер самого доходного источника, а главный (внешний) запрос сравнивает общий доход каждого жителя с полученным размером и, в зависимости от результата сравнения, либо добавляет жителя в таблицу результатов запроса, либо нет. Совокупно главный и вложенный запросы выражают исходный запрос и извлекают из БД требуемую информацию.

Вложенные SQL-запросы являются частью предложения WHERE или HAVING. В предложении WHERE они помогают отбирать отдельные строки, а в предложении HAVING — группы строк.

Вложенный запрос — это оператор SELECT, заключенный в круглые скобки. Однако между вложенным запросом и оператором SELECT имеется ряд отличий.

1. Таблица результатов вложенного запроса всегда состоит из одного столбца, т.е. список возвращаемых столбцов должен иметь только один элемент.
2. Во вложенный запрос не может входить предложение ORDER BY.
3. Вложенный запрос не может быть запросом на объединение нескольких различных операторов SELECT; допускается использование только одного оператора SELECT.
4. Имена столбцов, используемые во вложенном запросе, могут являться ссылками на столбцы таблиц главного (внешнего) запроса.

Чтобы проиллюстрировать последнее из перечисленных отличий, рассмотрим следующий запрос: определить адреса квартир, жители которых не могут оплачивать коммунальные услуги. Условно принимаем, что коммунальные услуги оплачиваются из расчета 60 руб. за квадратный метр (в год).

Таблица результатов формируется оператором:

```
SELECT ADR FROM FLAT
WHERE SKV*60>(SELECT SUM(SUMD) FROM PERSON
              WHERE FLAT.ADR = PERSON.ADR)
```

Столбец (или поле) FLAT.ADR во вложенном запросе является примером внешней ссылки. *Внешняя ссылка* представляет собой имя столбца, принадлежащего таблице, указанной в предложении FROM главного запроса, и не входящего ни в одну из таблиц, перечисленных в предложении FROM вложенного запроса.

Если во вложенном запросе имеется внешняя ссылка, то он называется связанным подзапросом. Особенностью связанного подзапроса является то, что он выполняется многократно, по одному разу для каждой строки таблицы, указанной в главном запросе.

Процедура выполнения связанного подзапроса состоит из следующих шагов:

1) выбрать строку из таблицы, имя которой указано в главном запросе; эту текущую строку назовем строкой-кандидатом (в примере это строка таблицы FLAT с полем ADR);

2) выполнить вложенный запрос с учетом значений, содержащихся в текущей строке-кандидате (в примере это значение поля FLAT.ADR, в соответствии с которым из таблицы PERSON выбираются сведения о жителях конкретной квартиры и вычисляется сумма их доходов);

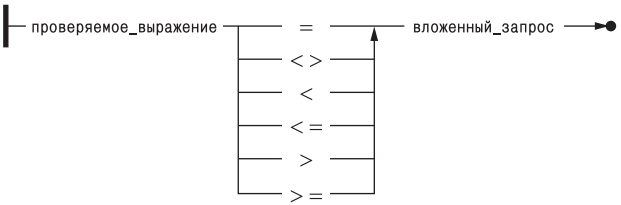
3) вычислить условие поиска главного запроса с учетом результатов вложенного запроса, выполненного на шаге 2; если вычислено значение TRUE, то текущая строка-кандидат включается в таблицу результатов;

4) повторять шаги 1—3 для следующей строки-кандидата, пока не будут проверены все строки таблицы, указанной в главном запросе.

Условия поиска с вложенным запросом

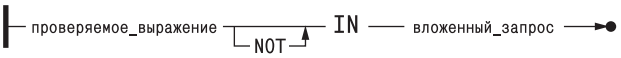
Вложенный запрос может являться частью условия поиска в предложении WHERE или HAVING. Возможны следующие условия поиска с вложенным запросом:

1. **Сравнение с результатом вложенного запроса.** Сравнивает значение выражения с одним значением, возвращенным вложенным запросом:



2. **Проверка на принадлежность результатам вложенного запроса.** Проверяет значение выражения на равенство с одним из значений множества, возвращенного вложенным запросом:

Пример запроса: вывести список квартир, в которых не установлены телефоны:




```
SELECT * FROM FLAT
WHERE ADR NOT IN (SELECT ADR FROM TPHONE)
```

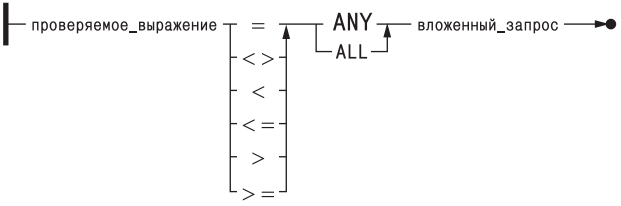
3. **Проверка на существование.** Проверяет наличие строк в таблице результатов вложенного запроса:

Пример запроса: вывести список жителей, имеющих источник дохода с размером большим 1000 руб.:

```
SELECT * FROM PERSON
WHERE EXISTS (SELECT * FROM PROFIT, HAVE_D
                WHERE PERSON.NOM = HAVE_D.NOM
                AND PROFIT.ID = HAVE_D.ID
                AND PROFIT. MONEY>1000)
```

Обратите внимание на то, что условие поиска EXISTS не использует результаты вложенного запроса. Проверяется только наличие результатов. По этой причине в SQL смягчается правило о единственности возвращаемого столбца во вложенном запросе и во вложенном запросе при проверке EXISTS допускается использование формы SELECT *.

4. **Многократное сравнение.** Сравнивает значение выражения с каждым из значений множества, возвращаемого вложенным запросом:



При проверке ANY проверяемое значение поочередно сравнивается с каждым значением, содержащимся в столбце, который сформирован вложенным запросом. Если любое из этих сравнений дает результат TRUE, то проверка ANY возвращает значение TRUE.

При проверке ALL проверяемое значение поочередно сравнивается с каждым значением, содержащимся в столбце данных, выбранных вложенным запросом. Если все сравнения дают результат TRUE, то проверка ALL возвращает значение TRUE.

3.4.11. Внесение изменений в базу данных

Язык SQL позволяет не только извлекать информацию из БД с помощью запросов на чтение, но и изменять содержащуюся в ней информацию с помощью запросов на добавление, удаление и обновление.

По сравнению с оператором SELECT, задающим запросы на чтение, операторы языка SQL, изменяющие содержимое БД, являются более простыми. Однако при изменении содержимого БД к СУБД предъявляется ряд дополнительных требований. При внесении изменений СУБД должна сохранять целостность данных и разрешать ввод в БД только допустимых значений, а также обеспечивать возможность одновременного изменения БД несколькими пользователями, чтобы они не мешали друг другу.

Добавление новых данных

Наименьшей единицей информации, которую можно добавить в реляционную БД, является одна строка. Существует два способа добавления новых строк в БД:

1) однострочный оператор INSERT позволяет добавить в таблицу новую строку (рис. 3.17);

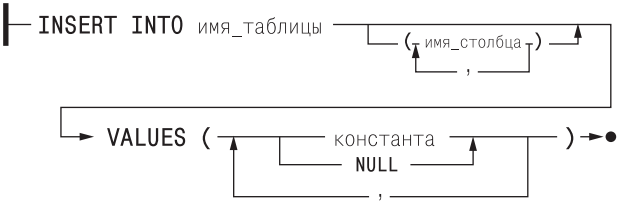


Рис. 3.17. **Синтаксическая диаграмма однострочного оператора INSERT**

2) многострочный оператор INSERT обеспечивает извлечение строк из одной части БД и добавление их в другую таблицу (рис. 3.18).

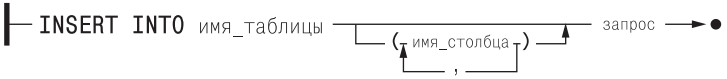


Рис. 3.18. **Синтаксическая диаграмма многострочного оператора INSERT**

В предложении INTO указывается целевая таблица, в которую добавляется новая строка, а в предложении VALUES содержатся значения данных для новой строки. Список столбцов определяет, какие значения в какой столбец заносятся.

Например, можно задать запросы для внесения в БД сведений о новой квартире, установке телефона в этой квартире и заселении этой квартиры новым жильцом:

а) INSERT INTO FLAT
VALUES (‘Зеленоград, 1001-45’, 40, 3,’H’)

б) INSERT INTO TPHONE (ADR, NTEL, TCATEGORY)
VALUES (‘Зеленоград, 1001-45’, ’999-1199’, ’C’)

в) INSERT INTO PERSON (NOM, FIO, RDATE, POL, ADR)
VALUES(14, ’Кукушкин Жан Ильич’, ’03/13/77’, ’M’, ’Зеленоград, 1001-45’)

Если список столбцов опущен, то последовательность значений данных должна точно соответствовать порядку столбцов в таблице (см. «*а*»).

В списке столбцов их имена могут указываться в любой удобной последовательности, определяющей затем порядок значений данных в предложении VALUES (см. «*б*»).

При добавлении в таблицу новой строки всем столбцам, имена которых отсутствуют в списке столбцов оператора INSERT, СУБД автоматически присваивает значение NULL либо значение, заданное по умолчанию при описании структуры таблицы (см. «*в*»), где отсутствует имя столбца SUMD).

Список столбцов в операторе INSERT служит, чтобы установить соответствие между значениями данных, содержащимися в предложении VALUES, и столбцами, для которых эти данные предназначены. Списки значений и столбцов должны содержать одинаковое число элементов, а тип данных каждого значения должен соответствовать типу соответствующего столбца.

Многострочный оператор INSERT добавляет в целевую таблицу одну или несколько строк. Источником новых строк служит запрос на чтение, содержащийся внутри оператора INSERT.

Например, можно добавить в таблицу PERSON сведения о новом жителе, у которого общий доход будет таким же, как у жителя с номером 16:

```
INSERT INTO PERSON
SELECT 20,'Алов Наум Нилыч','05/20/49','M', SUM
(MONEY),'Зеленоград, 1001-45' FROM HAVE_D, PROFIT
WHERE NOM = 16 AND HAVE_D.ID=PROFIT.ID
```

Такой способ определения общего дохода у жителя с номером 16, требующий осуществления соединения двух таблиц и использования агрегатной функции, обусловлен стандартом SQL1, который требует, чтобы имя целевой таблицы оператора INSERT не присутствовало в предложении FROM запроса, вложенного в него, т.е. запрещается добавление таблицы самой в себя.

В стандарте SQL2 это ограничение отсутствует, и «самодобавление» разрешено. Следовательно, рассмотренный пример можно модифицировать следующим образом:

```
INSERT INTO PERSON
SELECT 20, 'Алов Наум Нилыч','05/20/49',
'M', SUMD,'Зеленоград, 1001-45'
FROM PERSON WHERE NOM = 16
```

Если в БД имеется таблица OLD_PERSON для хранения сведений о пожилых гражданах, то данные в нее можно добавить из таблицы PERSON с помощью такого многострочного оператора INSERT:

```
INSERT INTO OLD_PERSON (NOM, FIO, RDATE, ADR)
SELECT NOM, FIO, RDATE, ADR FROM PERSON
WHERE RDATE < '01/01/1949'
```

Удаление существующих данных

Наименьшей единицей информации, которую можно удалить из реляционной БД, является строка. Удаление существующей строки из БД происходит, когда объект, представляемый этой строкой, перестает существовать в рассматриваемой предметной области. На примере учебной БД это выглядит следующим образом:

1) если житель лишается одного из своих источников дохода, то необходимо удалить строку из таблицы HAVE_D, содержащую номер жителя и идентификатор источника дохода, которого лишился житель;

2) если в квартире отключается телефон, то из таблицы TPHONE должна быть удалена строка, соответствующая этому телефону;

3) если квартира ликвидируется, необходимо удалить соответствующую строку в таблице FLAT; в случае, когда жители этой квартиры уезжают из Зеленограда, то их строки в таблице PERSON также должны быть удалены вместе с соответствующими строками в таблице HAVE_D; если жители переезжают в другую квартиру Зеленограда, то для них необходимо обновить значения в столбце ADR таблицы PERSON.

Для удаления выбранных строк из одной таблицы используется оператор DELETE (см. рис. 3.2).

В предложении FROM указывается таблица, содержащая удаляемые строки. В предложении WHERE указывается условие, которому должны соответствовать удаляемые строки.

Предположим, что необходимо зафиксировать в БД отключение телефона, установленного в квартире по адресу: Зеленоград, 1001-45:

```
DELETE FROM TPHONE
WHERE ADR ='Зеленоград, 1001-45'
```

Можно удалить несколько строк одним оператором DELETE, например, когда все жители одной квартиры уезжают из Зеленограда, то этот факт отражается в таблице PERSON удалением строк, соответствующих уезжающим жителям:

```
DELETE FROM PERSON
WHERE ADR ='Зеленоград, 1001-45'
```

Если предложение WHERE отсутствует в записи оператора DELETE, то такой оператор удаляет из таблицы все строки. Например, оператор DELETE FROM PERSON «опустошит» таблицу PERSON, удалив из нее все строки.

В предложении WHERE условие поиска может содержать вложенный запрос. Например, прежде чем удалять из таблицы PERSON информацию об уезжающих из города жителях некоторой квартиры, необходимо из таблицы HAVE_D удалить все строки, связанные с этими жителями и определяющие источники доходов этих жителей. Такое удаление обеспечивает следующий оператор:

```
DELETE FROM HAVE_D WHERE NOM IN
(SELECT NOM FROM PERSON WHERE
ADR ='Зеленоград, 1001-45')
```

Обновление существующих данных

Наименьшей единицей информации, которую можно обновить в реляционной БД является значение одного столбца в одной строке. Обновлять информацию требуется, когда изменяются значения атрибутов у объектов рассматриваемой предметной области. На примере учебной БД это выглядит следующим образом:

1) если в квартире изменяется номер установленного телефона, то соответствующей строке таблицы TPHONE должно быть обновлено значение столбца NTEL;

2) если жители одной квартиры переезжают в другую квартиру, то столбец ADR таблицы PERSON для этих жителей необходимо обновить, чтобы учесть адрес их нового места жительства.

Для обновления значения одного или нескольких столбцов в выбранных строках одной таблицы предназначен оператор UPDATE (рис. 3.19).

Рис. 3.19. **Синтаксическая диаграмма оператора UPDATE**

В операторе указывается целевая таблица, которая должна быть модифицирована. Предложение WHERE отбирает строки таблицы, подлежащие обновлению. В предложении SET указывается, какие столбцы в выбранных строках таблицы должны быть обновлены, и для них задаются новые значения.

Следующий оператор UPDATE изменяет номер и категорию телефона, установленного в квартире с заданным адресом:

```
UPDATE TPHONE SET NTEL = '111-0000', TCATEGORY = 'Д'
WHERE ADR = 'Зеленоград, 1001-45'
```

Оператор UPDATE может одновременно обновить столбцы в нескольких строках, соответствующих условию поиска. Например, можно изменить адрес проживания всех жителей одной квартиры:

```
UPDATE PERSON SET ADR = 'Зеленоград, 1801-12'
WHERE ADR = 'Зеленоград, 1001-45'
```

Этот оператор выполняется следующим образом: все строки таблицы PERSON по очереди проверяются на соответствие

условию поиска. Строки, для которых условие поиска имеет значение TRUE, обновляются, а строки, для которых условие поиска имеет значение FALSE или NULL, не обновляются.

Выражение в операции присваивания может быть любым правильным выражением языка SQL, результирующее значение которого имеет тип данных, соответствующий целевому столбцу.

Значение выражения вычисляется на основе значений строки, которая в данный момент обновляется в целевой таблице. Выражение не может включать в себя какие-либо агрегатные функции или запросы.

Если выражение содержит ссылку на один из столбцов целевой таблицы, то для вычисления выражения используется значение этого столбца в текущей строке, которое было перед обновлением. То же самое справедливо для ссылок на столбцы в предложении WHERE. Например:

```
UPDATE FLAT SET NROOMS = 5,
SKV = NROOMS + 1 WHERE NROOMS = 3
```

Для строк таблицы FLAT, в которых столбец NROOMS содержит значение, равное 3, оператор UPDATE запишет в столбец NROOMS значение 5, а в столбец SKV — значение 4 (3 + 1 !). Таким образом, порядок операций присваивания в предложении SET не влияет на результат.

Если предложение WHERE отсутствует в записи оператора UPDATE, то обновляются все строки целевой таблицы, например:

```
UPDATE PERSON SET SUMD = SUMD/1000
```

Предложение WHERE может содержать вложенный запрос, если необходимо отбирать строки на обновление, опираясь на информацию из других таблиц. Например, можно увеличить общий доход жителей, имеющих не более одного источника дохода; выплатив единовременно по 500 руб. каждому из них:

```
UPDATE PERSON SET SUMD=SUMD+500
 WHERE 1 >= (SELECT COUNT (*) FROM HAVE_D
 WHERE PERSON.NOM = HAVE_D.NOM)
```

3.4.12. Создание базы данных

Операторы SELECT и INSERT, DELETE, UPDATE, рассмотренные ранее, предназначены для обработки данных и входят в те части языка SQL, которые называются соответственно языками запросов данных DQL (Data Query Language) и обработки данных DML (Data Manipulation Language).

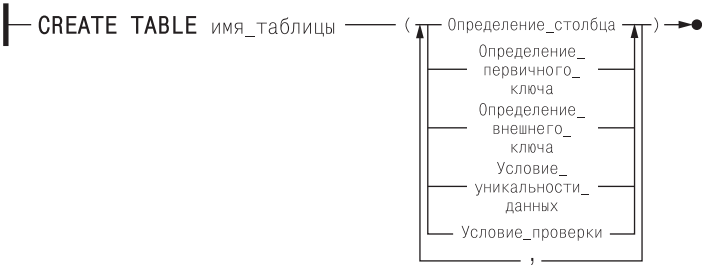
Создание БД заключается в описании таблиц, образующих реляционную БД, с указанием существующих связей между таблицами.

Для описания новой таблицы предназначен оператор CREATE TABLE (рис. 3.20), который определяет новую таблицу и подготавливает ее к приему данных. Этот оператор относится к языку определения данных DDL (Data Definition Language), предназначенному для изменения структуры БД.

Посмотрим, как описать таблицы, входящие в приведенную на рис. 2.11 учебную БД (указаны типы данных, используемые в СУБД InterBase):

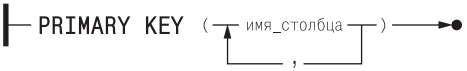
```
CREATE TABLE FLAT
   (ADR CHAR (30) NOT NULL,
   SKV FLOAT DEFAULT 0.00,
   NROOMS SMALLINT DEFAULT 0,
   KCATEGORY CHAR (1) DEFAULT 'H',
   PRIMARY KEY (ADR),
   CHECK (NROOMS BETWEEN 0 AND 4),
   CHECK (KCATEGORY IN ('П',' H',' K')))

CREATE TABLE PROFIT
   (ID INTEGER NOT NULL,
   SOURCE CHAR (20) NOT NULL,
   MONEY FLOAT DEFAULT 0.00,
   PRIMARY KEY (ID),
   CHECK (MONEY>=0) )

CREATE TABLE TPHONE
   (NTEL CHAR (8) NOT NULL,
   TCATEGORY CHAR (1) DEFAULT 'O',
   ADR CHAR (30),
   PRIMARY KEY (NTEL),
```

```
FOREIGN KEY (ADR) REFERENCES FLAT,
CHECK (NTEL LIKE '_ _ _-_ _ _ _'),
CHECK (TCATEGORY IN ('О',' Д','С')))
```

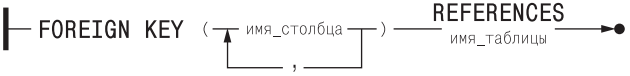


Определение_столбца

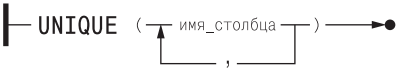
Определение_первичного_ключа



Определение_внешнего_ключа



Условие_уникальности_данных



Условие_проверки

Рис. 3.20. **Синтаксическая диаграмма оператора CREATE TABLE**

```
CREATE TABLE PERSON
   (NOM INTEGER NOT NULL,
   FIO CHAR (30) NOT NULL,
   RDATE DATE NOT NULL,
   POL CHAR (1),
   SUMD FLOAT DEFAULT 0.00,
   ADR CHAR (30),
   PRIMARY KEY (NOM),
   FOREIGN KEY (ADR) REFERENCES FLAT,
   CHECK (POL IN ('М','Ж')))

CREATE TABLE HAVE_D
   (NOM INTEGER NOT NULL,
   ID INTEGER NOT NULL,
   PRIMARY KEY (NOM, ID),
   FOREIGN KEY (NOM) REFERENCES PERSON,
   FOREIGN KEY (ID) REFERENCES PROFIT)
```

3.5. Язык запросов по образцу QBE

Запросы на выборку данных являются самым распространенным типом запросов. Они предназначены для извлечения данных из одной или нескольких таблиц и отображения полученных результатов в виде сетки с выбранными данными. В сетке с выбранными данными извлеченная из таблицы информация отображается в виде набора столбцов и строк, подобно обычной электронной таблице. Запросы на выборку допускают группирование записей, а также вычисление сумм, счетчиков, средних значений и применение агрегатных функций других типов.

В современных СУБД широко используются табличные языки запросов. Наиболее распространенным среди них является язык QBE (Query-By-Example — запрос по образцу (примеру)). Язык QBE предназначен для работы в интерактивном режиме и ориентирован на конечного пользователя. Он реализован во многих современных СУБД, например в dBase IV и более старших версиях этой системы, Paradox, Access. Конкретные реализации этого языка несколько отличаются друг от друга, но все они построены по единому принципу.

Подход, воплощенный в языке QBE, заключается в следующем [6]. В окне формирования запроса, называемом окном Query, выделяются две зоны. В первой из них высвечивается структура таблицы, данные из которой будут участвовать в запросе. В качестве исходных для запроса могут указываться не только таблицы из БД, но и другие запросы.

Во второй зоне (форме запроса табличной формы) пользователь задает условия запроса. В этой зоне пользователь определяет, какие поля (столбцы) участвуют в формировании запроса, а также условия отбора и некоторые другие характеристики запроса. Например, если пользователю необходимо получить все строки с заданным значением конкретного атрибута, то в соответствующем столбце формы запроса указывается это значение.

В общем случае формирование запроса начинается с выбора таблиц, для которых предназначен запрос. Далее во второй зоне окна Query задаются отметки (в виде галочек) для указания полей, включаемых в таблицу ответов ANSWER. Для сужения области действия запроса в нем указываются условия, которым должны соответствовать выбираемые данные. Когда запрос сформирован, можно выполнить его нажатием соответствующей клавиши (например, Run Query). СУБД обрабатывает запрос, преобразуя его в SQL-оператор, после выполнения которого результаты отображаются в таблице ANSWER (рис. 3.21).

Чтобы включить конкретное поле в таблицу ответов, нужно в окне Query поставить отметку в этом поле. Если установить курсор мыши на маленьком окошке в интересующем поле и нажать левую кнопку мыши, то при использовании СУБД Paradox появится список возможных отметок:

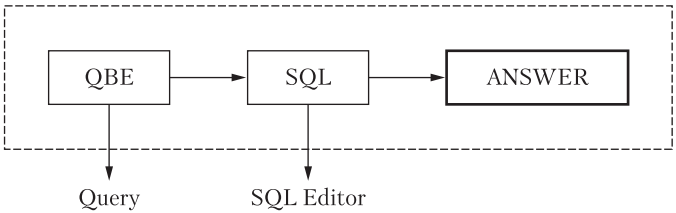


Рис. 3.21. **Процесс выполнения QBE-запроса**

v — выбор неповторяющихся строк и сортировка в порядке возрастания;

— отмена выбора;

v+ — выбор всех строк без сортировки;

v↓ — выбор неповторяющихся строк и сортировка в порядке убывания.

Например, чтобы определить, какие различные категории квартир имеются, нужно поставить отметку в поле KCategory, а чтобы вывести всю таблицу FLAT, нужно отметить все поля таблицы с помощью универсальной кнопки, расположенной в первой колонке запроса под именем таблицы.

Рис. 3.22. **Структура таблицы FLAT и форма QBE-запроса**

Чтобы выбрать из таблицы данные, удовлетворяющие некоторым условиям, нужно в запросе для отдельных полей справа от окошка записать условия, относящиеся к этим полям. Например, на рис. 3.22 показан запрос на языке QBE для СУБД Paradox, выбирающий из таблицы FLAT сведения о приватизированных (П) или (ОК) коммунальных (К) квартирах с площадью в диапазоне от 20 до 40 м^2, расположенных в 8-м микрорайоне г. Зеленограда. Запятая в условии для поля Skv обозначает логическую операцию И.

Кроме задания условия отбора данных, при описании запроса язык QBE дает возможность указать, какие атрибуты и в какой последовательности входят в таблицу ответов. В нее могут помещаться не только реальные поля, которые хранятся в одной из исходных таблиц, но и вычисляемые поля.

Есть два вида вычислений, которые могут выполняться в запросах, формах и отчетах: это агрегирующие операторы, которые выполняют операции над группой строк, и обычные вычисления, затрагивающие отдельные поля одной или нескольких связанных строк.

Набор агрегатных функций может быть различным в разных системах. Обычно имеются следующие функции: Sum (сумма), Min (минимум), Max (максимум), Avg (среднее), Count (подсчет). Некоторые системы включают такие до-

полнительные статистические функции, как отклонение, стандартное отклонение, дисперсия и т.д.

Использование агрегатных функций предполагает, что таблица упорядочена по тому полю (полям), по которому ведется агрегирование. Некоторые СУБД сами автоматически выполняют упорядочение данных по необходимым полям, другие — нет. В последнем случае, если пользователь не задаст правильно требуемое упорядочение, результат будет искаженным.

При использовании СУБД Paradox значение поля в таблице ответов вычисляется с помощью оператора CALC, записываемого в поле справа от окошка. Вычисляемое выражение может содержать элемент образца — переменную, принимающую текущее значение поля данных. Элемент образца обозначается именем, перед которым набирается символ подчеркивания (символ подчеркивания в запросе не отображается, а элемент образца выделяется особым цветом). Элемент образца и оператор CALC разделяются запятой, например: _a, CALC _a+2.

Результаты вычислений, выводящиеся в поле, не запоминаются в исходной таблице. Вместо этого вычисления снова проводятся всякий раз, когда выполняется запрос, поэтому результаты всегда представляют текущее содержимое базы данных. Обновить вычисленные результаты вручную невозможно (таблица, содержащая вычисляемое поле, имеет статус «только для чтения»).

Структуры всех таблиц, которые нужны для реализации запроса должны быть вызваны на экран.

Дальнейшие действия, которые необходимо выполнить, чтобы осуществить соединение таблиц, зависят от используемой СУБД. Так, в некоторых системах для соединения таблиц используются упомянутые выше элементы образца. Их имена могут быть любыми, но они должны быть одинаковыми в обеих соединяемых таблицах (рис. 3.23).

Рис. 3.23. **Использование элементов образца с именем join1 при соединении таблиц в QBE-запросе для СУБД Paradox**

В других СУБД используются визуальные способы установления связей между таблицами: для связывания таблиц следует мышью позиционироваться на нужном поле в главной таблице и, не отпуская кнопки мыши, переместиться к полю в подчиненной таблице. На экране появится линия, связывающая таблицы. Существуют и другие способы установления связей.

Теоретически возможны разные типы соединений таблиц. Наиболее распространенным является соединение, при котором в таблицу ответов помещаются те соединенные строки, для которых значение поля связи главной таблицы совпадает с соответствующим полем в подчиненной таблице. В описанных выше случаях устанавливается именно такое соединение. Различают «левое» и «правое» соединения, когда в таблицу ответов помещаются все строки из главной или подчиненной таблицы соответственно, даже если для них нет связанных строк в другой таблице. Но не все системы позволяют с помощью языка QBE реализовывать такие соединения. В случаях, когда возможно задание разных типов соединений, конкретный способ реализации отличается в разных СУБД. Так, в Access «левое» и «правое» соединения можно определить, задав для связи таблиц «параметры объединения» или использовав средства языка SQL.

Работа с несколькими таблицами в конкретных СУБД различается не только тем, каким способом можно определить связь между таблицами. Так, например, одни системы обязывают пользователя связать те таблицы, которые указываются как исходные для запроса; другие — автоматически связывают открытые таблицы по тем полям, которые система воспринимает как поля связи (чаще всего это поля, имеющие одинаковые имена, тип и длину); третьи — оставляют эти таблицы изолированными, если пользователь не указал, как они должны быть связаны, четвертые — выполняют декартово произведение открытых таблиц. Например в Access, если таблицы не связаны, то при выполнении запроса это приводит к связыванию каждой строки одной таблицы с каждой строкой другой.

Кроме собственно поисковых запросов язык QBE позволяет выполнять и другие операции, например корректировку данных. Набор допустимых операций, а также способы их задания несколько различаются в разных системах.

В языке QBE, реализованном СУБД Paradox, для изменения значения поля предназначен оператор CHANGETO,

причем при использовании этого оператора все поля должны быть неотмеченными. Например, на рис. 3.24 показан запрос, переводящий все однокомнатные квартиры (NRooms = 1) в категорию приватизированных.

Рис. 3.24. **QBE-запрос, вносящий изменения в таблицу базы данных**

Для вставки или удаления строк (в таблицу или из таблицы) нужно нажать кнопку мыши, когда курсор установлен в первой колонке запроса под именем таблицы. Из появившегося меню выбирается команда INSERT или DELETE. Для INSERT в окне Query набираются значения полей вставляемой строки и нажимается кнопка Run Query. Для DELETE в окне Query набираются условия, которым должны удовлетворять значения полей удаляемых строк, и нажимается кнопка Run Query. При использовании команд INSERT и DELETE все поля должны быть неотмеченными.

Для удобства восприятия результатов язык QBE позволяет задать упорядоченность данных в таблице ответов. Возможности задания упорядочения различаются в разных СУБД: некоторые системы разрешают проводить упорядочение по произвольным полям, другие требуют, чтобы поле упорядочения стояло в таблице ответов обязательно первым, а если упорядочение ведется по нескольким полям, то чтобы эти поля следовали в таблице ответов друг за другом в порядке их расположения в таблице базы данных; некоторые СУБД различают обычное и словарное упорядочение (когда учитывается и не учитывается регистр соответственно), другие — нет; в некоторых системах, даже если не задано никакое упорядочение, таблица ответов всегда выдается упорядоченной по первому полю таблицы ответов и т.п. Запросы на языке QBE могут быть сохранены для последующего многократного использования.

Глава 4

РЕЛЯЦИОННЫЕ СУБД

Эффективная коллективная работа с информацией невозможна без использования общей БД, устанавливаемой в компьютерной сети на специально выделенном компьютере — сервере, к которому имеют одновременный доступ компьютеры пользователей. На сервере устанавливается и СУБД, контролирующая доступ клиентов к БД и называемая SQL-сервером, если языком взаимодействия с СУБД является язык SQL.

Для информационных систем с клиент-серверной архитектурой характерна максимальная разгрузка клиента от вычислительной работы, которая переносится на сервер, и существенное улучшение защищенности данных от несанкционированного доступа или ошибочных изменений. Для реализации клиент-серверной архитектуры применяются так называемые промышленные SQL-серверы, например, InterBase, Oracle, SQL Server, Informix, Sybase Adaptive Server, DB2.

Типичная СУБД, в том числе и реляционная СУБД, реализует ряд важных функций [13], рассматриваемых далее.

4.1. Функции СУБД

СУБД должна обеспечивать *хранение, извлечение и обновление данных* в базе. Это самая фундаментальная функция СУБД, которая реализуется таким способом, чтобы скрывать от конечного пользователя внутренние детали устройства системы (например, файловую организацию или используемые структуры хранения).

СУБД должна поддерживать *доступный конечным пользователям каталог*, в котором хранится описание элементов данных. Ключевой особенностью архитектуры ANSI-SPARC

(см. разд. 1.3) является наличие интегрированного *системного каталога* с данными о схемах, пользователях, приложениях и т.д. Предполагается, что каталог доступен как пользователям, так и функциям СУБД. Системный каталог (или словарь данных) является хранилищем информации, описывающей данные в базе данных (по сути, это «данные о данных», или метаданные). Обычно в системном каталоге хранятся следующие сведения:

— имена, типы и размеры элементов данных;

— имена связей;

— накладываемые на данные ограничения поддержки целостности;

— имена пользователей, которым предоставлено право доступа к данным;

— внешняя, концептуальная и внутренняя схемы и отображения между ними;

— статистические данные, например частота транзакций и счетчики обращений к объектам базы данных.

СУБД должна обеспечивать *поддержку транзакций*, гарантирующую выполнение либо всех операций обновления данной транзакции, либо ни одной из них. *Транзакция* представляет собой набор действий, выполняемых отдельным пользователем или прикладной программой с целью доступа или изменения содержимого базы данных. Если во время выполнения транзакции произойдет сбой, например из-за выхода из строя компьютера, целостность базы данных будет нарушена, поскольку некоторые изменения уже будут внесены, а остальные — еще нет. Поэтому все частичные изменения должны быть отменены для возвращения базы данных в прежнее состояние.

СУБД должна *управлять параллельной работой* пользователей с базой данных, гарантируя корректное обновление базы данных при одновременном выполнении операций обновления многими пользователями. Одна из основных целей создания и использования СУБД заключается в том, чтобы множество пользователей могло осуществлять параллельный доступ к совместно обрабатываемым данным. Параллельный доступ сравнительно просто организовать, если все пользователи выполняют только чтение данных, поскольку в этом случае они не могут помешать друг другу. Однако, когда два или больше пользователей одновременно получают доступ к базе данных, конфликт с нежелатель-

ными последствиями легко может возникнуть, если хотя бы один из них попытается обновить данные.

СУБД должна предоставлять средства *восстановления базы данных* при ее повреждении или разрушении. При сбое транзакции база данных должна быть возвращена в прежнее, непротиворечивое состояние. Подобный сбой может произойти в результате выхода из строя запоминающего устройства, ошибки аппаратного или программного обеспечения, которые могут привести к останову СУБД. Кроме того, пользователь может обнаружить ошибку во время выполнения транзакции и потребовать ее отмены. Во всех этих случаях СУБД должна предоставить возможность восстановления базы данных и возврата ее к непротиворечивому состоянию.

СУБД должна *контролировать доступ к данным*, чтобы гарантировать получение или изменение данных только для пользователей с соответствующими правами. Термин «безопасность» относится к защите базы данных от преднамеренного или случайного несанкционированного доступа. СУБД обеспечивает механизмы подобной защиты данных.

Любая СУБД должна *поддерживать обмен данными*, легко взаимодействуя с различными существующими диспетчерами обмена данными. Даже СУБД для персональных компьютеров должны поддерживать работу в локальной сети, чтобы вместо нескольких разрозненных баз данных для каждого отдельного пользователя можно было бы установить одну централизованную базу данных и использовать ее как общий ресурс для всех заинтересованных пользователей.

СУБД должна *поддерживать целостность данных*, контролируя соответствие данных и их изменений заданным правилам. Целостность базы данных означает корректность и непротиворечивость хранимых данных и может рассматриваться как еще один тип защиты базы данных. Помимо того, что данный вопрос связан с обеспечением безопасности, он имеет более широкий смысл, поскольку целостность связана с качеством самих данных. Целостность обычно выражается в виде ограничений или правил сохранения непротиворечивости данных, которые не должны нарушаться в базе (см. разд. 6.1).

СУБД должна *поддерживать независимость* программ *от* фактической структуры базы *данных*. Независимость от

данных обычно достигается за счет реализации механизма поддержки представлений или подсхем.

СУБД должна предоставлять также некоторый набор различных *вспомогательных функций*, предназначенных для оказания помощи администратору баз данных в эффективном администрировании БД. Одни функции необходимы на внешнем уровне, а потому они могут быть запрограммированы самим администратором баз данных, тогда как другие реализуются на внутреннем уровне системы и потому должны быть предоставлены разработчиком СУБД.

4.2. Microsoft Access

Microsoft Access — наиболее широко используемая в среде Windows реляционная СУБД, типичная для персональных компьютеров и обеспечивающая хранение, сортировку и поиск данных для множества приложений.

В СУБД Access для создания таблиц, запросов, форм и отчетов предусмотрен графический интерфейс пользователя; для разработки настраиваемых приложений с базой данных есть инструментальные средства, использующие макроязык Microsoft Access или язык VBA (Microsoft Visual Basic for Applications). Кроме того, в СУБД Access предусмотрены программы, называемые мастерами (Wizards), которые упрощают многие из процессов формирования приложений с базой данных, проводя пользователя через ряд диалоговых окон в запросно-ответном режиме. В СУБД Access предусмотрены также конструкторы (Builders), которые могут помочь пользователю сформировать синтаксически правильные выражения, например операторы языка SQL. СУБД Access поддерживает значительную часть стандарта языка SQL.

База данных хранится в одном файле (*.mdb). Кроме таблиц базы данных, в этом же файле сохраняются компоненты приложения для работы с базой данных — экранные формы, отчеты, запросы, программные модули. Работу с базой данных обеспечивает машина баз данных, которая используется для доступа к данным.

В табл. 4.1 приведены сведения о типах данных, которые могут иметь поля в таблицах базы данных.

Таблица 4.1

Типы данных СУБД Access

Тип данных полей	Тип данных в VBA	Использование	Размер
Текстовый	String	Текст, состоящий из любых символов в кодировке Unicode (2 байта на символ)	До 255 символов
Поле MEMO	String	Текст в кодировке Unicode	До 64 000 символов
Числовой (Байт, Целое, Длинное целое, Одинарное с плавающей точкой, Двойное с плавающей точкой)	Byte, Integer, Long, Single, Double	Числовые данные	1, 2, 4 или 8 байтов
Дата/время: Полный формат даты, Длинный формат даты, Средний формат даты, Краткий формат даты, Длинный формат времени, Средний формат времени, Краткий формат времени	Date	Дата и время: 31.12.04 23:55:59 31 декабря 2004 г. 31-дек-04 31.12.04 23:55:59 11:55 23:55	8 байтов (при активации поля всегда показывает полный формат даты)
Денежный	Currency	Значения валют. Денежный тип используется для предотвращения округлений во время вычислений. Предполагает до 15 символов в целой части числа и 4 — в дробной	8 байтов
Счетчик		Автоматическая вставка последовательных (увеличивающихся на 1) или случайных чисел при добавлении записи	4 байта. 16 байтов только для кодов репликации
Логический	Boolean	Поля, содержащие только одно из двух возможных значений, таких как Да/Нет, Истина/Ложь, Вкл/Выкл	1 бит

Окончание табл. 4.1

Тип данных полей	Тип данных в VBA	Использование	Размер
Поле объекта OLE	String	Объекты (например, документы Microsoft Word, электронные таблицы Microsoft Excel, рисунки, звуки и другие двоичные данные), созданные в программах, использующих протокол OLE. Объекты могут быть связанными или внедренными	До 1 гигабайта (ограничено объемом диска)
Гиперссылка	String	Поле, в котором хранятся гиперссылки. Гиперссылка может иметь вид URL-адреса	До 64 000 символов

СУБД Access может использоваться как автономная система на одном персональном компьютере или как многопользовательская система в сети. Начиная с СУБД Access 2000 предоставляется выбор из двух машин баз данных (data engines): первоначальной версии машины баз данных Jet и новой Microsoft Data Engine (MSDE), которая совместима с Microsoft BackOffice SQL Server (продуктом компании Microsoft для администрирования локальных корпоративных сетей) [13].

В основе MSDE лежит та же машина базы данных, что и в СУБД Microsoft SQL Server, предоставляющая пользователям возможность писать масштабируемые приложения на компьютере с системой Windows 95, которые затем можно перенести в высокопроизводительные многопроцессорные кластеры (группы компьютеров), работающие под управлением системы Windows 2000 Server. Машина MSDE предоставляет также процедуру преобразования данных, позволяющую пользователям впоследствии наращивать вычислительные возможности до уровня Microsoft SQL Server, но MSDE в отличие от Microsoft SQL Server имеет ограничение на размер базы данных в 2 гигабайта.

Физическая модель базы данных, с которой работает СУБД Access, как и Microsoft SQL Server, делит данные, хранящиеся в ее табличных структурах, на страницы данных размером в 2 килобайта, что соответствует размеру

стандартного кластера файла жесткого диска в DOS. Каждая страница содержит одну или несколько записей. Запись не может занимать больше одной страницы, хотя записи Memo (поля примечаний) и поля объектов OLE могут храниться на отдельных страницах. СУБД Access использует в качестве стандартного способа хранения записи переменной длины и упорядочивает записи с помощью индекса первичного ключа. При использовании формата хранения записей с переменной длиной каждая запись занимает только пространство, необходимое для хранения ее фактических данных.

Основные характеристики системы при работе с базами данных приведены в табл. 4.2.

Таблица 4.2

Основные характеристики Microsoft Office Access 2003

Характеристика	Максимальное значение
База данных	
Размер файла базы данных (*.mdb)	2 гигабайта за вычетом места, необходимого системным объектам
Число объектов в базе данных	32 768
Модули (включая формы и отчеты, свойство Наличие модуля (HasModule) которых имеет значение True)	1 000
Число знаков в имени объекта	64
Число знаков в пароле	14
Число знаков в имени пользователя или имени группы	20
Число одновременно работающих пользователей	255
Таблица	
Число знаков в имени таблицы	64
Число знаков в имени поля	64
Число полей в таблице	255
Число открытых таблиц	2048 (фактическое число может быть меньше из-за внутренних таблиц, открываемых Microsoft Access)

Продолжение табл. 4.2

Характеристика	Максимальное значение
Размер таблицы	2 гигабайта за вычетом места, необходимого системным объектам
Число знаков в текстовом поле	255
Число знаков в поле MEMO	65 535 при вводе данных через интерфейс пользователя;1 гигабайт для хранения знаков при программном вводе данных
Размер поля объекта OLE	1 гигабайт
Число индексов в таблице	32
Число полей в индексе	10
Число знаков в сообщении об ошибке	255
Число знаков в условии на значение записи	2048
Число знаков в описании таблицы или поля	255
Число знаков в записи (кроме полей MEMO и полей объектов OLE)	2000
Число знаков в значении свойства поля	255
Запрос	
Число установленных связей	32 на одну таблицу за вычетом числа индексов, находящихся в таблице для полей или сочетаний полей, которые не участвуют в связях
Число таблиц в запросе	32
Число полей в наборе записей	255
Размер набора записей	1 гигабайт
Предел сортировки	255 знаков в одном или нескольких полях
Число уровней вложения запросов	50
Число знаков в ячейке на бланке запроса	1024
Число знаков для параметра в запросе с параметрами	255
Число операторов AND в предложении WHERE или HAVING	99
Число знаков в инструкции SQL	приблизительно 64 000
Форма и отчет	
Число знаков в надписи	2048
Число знаков в поле	65 535

Окончание табл. 4.2

Характеристика	Максимальное значение
Ширина формы или отчета	22 дюйма (55,87 см)
Высота раздела	22 дюйма (55,87 см)
Высота всех разделов плюс заголовки разделов (в режиме конструктора)	200 дюймов (508 см)
Число уровней вложения форм или отчетов	7
Число полей или выражений, которые можно отсортировать или сгруппировать в отчете	10
Число заголовков и примечаний в отчете	1 заголовок/примечание отчета; 1 заголовок/примечание страницы; 10 заголовков/примечаний групп
Число печатных страниц в отчете	65 536
Число элементов управления и разделов, которые можно добавить за время существования формы или отчета	754
Число знаков в инструкции SQL, работающей в качестве свойства Источник записей (RecordSource) или Источник строк (RowSource) формы, отчета или элемента управления (оба .mdb и .adp)	32 750
Макрос	
Число макрокоманд в макросе	999
Число знаков в условии	255
Число знаков в комментарии	255
Число знаков в аргументе макрокоманды	255

4.3. Microsoft SQL Server

Microsoft SQL Server — одна из наиболее мощных СУБД, имеющая клиент-серверную архитектуру [1, 15]. Особенностью СУБД является возможность ее работы только в операционных системах ряда Microsoft Windows NT (NT Server 4.0, 2000 Professional, 2000 Server, Server 2003) и Windows XP Professional; при этом клиентская часть может взаимодействовать с СУБД, работая под управлением Microsoft Windows 98 и других операционных систем.

В своем составе система имеет средства создания баз данных, работы с информацией баз данных, перенесения данных

из других систем и в другие системы, резервного копирования и восстановления данных, развитую систему транзакций, систему репликации данных, реляционную подсистему для анализа, оптимизации и выполнения запросов клиентов, систему безопасности для управления правами доступа к объектам базы данных и т.п. Система не содержит средств разработки клиентских приложений.

Для правильного проектирования баз данных необходимо знание типов данных, которые могут использоваться для полей таблиц в базе. В табл. 4.3 приведены типы данных в системе SQL Server с разбивкой их на группы по видам.

Таблица 4.3

Типы данных СУБД Microsoft SQL Server

Тип	Описание
Двоичные данные:	
binary [(*n*)]	максимальная длина 8000 байт (*n*)
varbinary [(*n*)]	данные переменной длины, максимальная длина 8000 байт (*n*)
image	максимальная длина 2 147 483 647 байт
bit	тип данных, который принимает значения 1 или 0
Символьные данные:	
char [(*n*)]	максимальная длина 8000 символов (*n*)
varchar [(*n*)]	тип переменной длины, максимально 8000 символов (*n*)
text	максимальная длина 1 073 741 823 символов
Символьные данные в кодировке Unicode:	
nchar (*n*)	максимальная длина 4000 символов (*n*)
nvarchar (*n*)	переменной длины в кодировке Unicode максимальная длина 4000 символов (*n*)
ntext	максимальная длина 1 073 741 823 символов
Числовые целые данные:	
bigint	диапазон от –922 337 203 685 4775808 до 922 337 203 685 4775807
Int	диапазон от –2 147 483 648 до 2 147 483 647
smallint	диапазон от –32 768 до 32 767
tinyint	диапазон от 0 до 255
Числовые данные с дробной частью числа:	
decimal[(*p*[, *s*])]	диапазон от $-10^{38}-1$ до $10^{38}-1$ с задание фиксированного количества знаков (*p* — всего и *s* — в дробной части), максимальное общее количество знаков 38
numeric	то же, что и decimal
float [(*n*)]	диапазон от $+2.29 \cdot 10^{-308}$ до $+1.79 \cdot 10^{308}$

Окончание табл. 4.3

Тип	Описание
real	числа с 7-значной точностью в диапазоне от $+1.18 \cdot 10^{-38}$ до $+3.40 \cdot 10^{38}$
Тип дата и время:	
datetime	диапазон от 1.01.1753 до 31.12.9999 с точностью 3.33 мс
smalldatetime	диапазон от 1.01.1900 до 6.06.2079 с точностью 1 мин.
Денежный тип:	
money	диапазон от – 922 337 203 685 477.5808 до +922 337 203 685 477.5807
smallmoney	диапазон от –214 748.3648 до +214 748.3647
Данные специальных типов:	
timestamp	счетчик, автоматически увеличивающийся, имеющий уникальное значение для базы данных (тип binary(8) или varbinary(8))
uniqueidentifier	тип, который содержит уникальный идентификационный номер (GUID), сохраняемый как 16-битная двоичная строка
sql_variant	тип, который сохраняет значения различных типов, кроме text, ntext, timestamp и sql_variant
sysname	тип — синоним nvarchar, используется для ссылок на имена объектов базы данных

В табл. 4.4 приведены некоторые характеристики системы.

Таблица 4.4

Некоторые характеристики Microsoft SQL Server

Характеристика	Максимальное значение
Размер базы данных	1 048 516 терабайт
Количество объектов в базе данных	2 147 483 647
Количество экземпляров сервера на одном компьютере	16
Количество баз данных в одном экземпляре сервера	32767
Количество файлов в базе данных	32767
Количество таблиц в базе данных	Ограничено количеством объектов в базе
Количество полей в таблице базы	1024
Размер файла данных	32 терабайт
Длина идентификаторов	128 символов
Уровень вложенных процедур	32
Уровень вложенных запросов	32
Количество некластерных индексов для одной таблицы базы	249

Окончание табл. 4.4

Характеристика	Максимальное значение
Количество полей в одном индексе	16
Количество байт в одном индексе	800
Количество таблиц в одном запросе	256
Количество байт в одной строке таблицы	8060

SQL Server имеет входной язык под названием Transact-SQL, в котором, помимо базовых SQL-операторов, предусмотрены программные конструкции — параметры, переменные и логические структуры (IF, WHILE и т.д.). Реализация логики приложения в SQL Server возможна тремя способами: посредством процедур на Transact-SQL и хранимых запросов, которые вызываются с помощью программы SQL Query Analyzer, посредством хранимых процедур, которые вызываются из прикладных программ или через SQL Query Analyzer, и посредством триггеров, которые вызываются SQL Server при выполнении определенных действий с базой данных.

Поведение SQL Server при управлении параллельной обработкой определяется тремя факторами: уровнем изоляции транзакции, поведением курсора и блокировочными подсказками, указанными пользователем в предложении SELECT. Имея блокировочные подсказки, указанные разработчиком, SQL Server самостоятельно налагает блокировки.

SQL Server поддерживает создание резервных копий журналов, а также полных и дифференциальных резервных копий базы данных. Предусмотрены три модели восстановления: простая, полная и выборочная. При простой модели журнал не ведется и записи из журнала не обрабатываются. При полной модели все операции, выполняемые с базой данных, заносятся в журнал и затем воспроизводятся при восстановлении. В случае выборочной модели при ведении журнала опускаются определенные виды транзакций, для записи которых требуется много места.

4.4. Oracle

Oracle — мощная и устойчивая СУБД, способная работать с базами данных под управлением различных операционных систем, включая Windows 98, Windows 2000, Windows XP,

несколько вариантов Unix, ряд операционных систем для больших ЭВМ и Linux [15]. Она имеет длительную историю разработки и использования и является самой популярной СУБД в мире, о чем свидетельствует отчет компании International Data Company (IDC) за май 2006 г.: в общем объеме продаж программного обеспечения систем хранения данных Oracle занимала 44,6%, Microsoft — только 16,8%. Значительная часть технологии Oracle открыта для разработчика, что обеспечивает большую гибкость при ее конфигурировании и настройке.

Однако все это означает, что Oracle может быть сложной в установке, и для работы с ней необходима высокая квалификацифия. Более того, методики, которые работают в версии Oracle, предназначенного для одной операционной системы, могут потребовать модификации в версии для другой операционной системы.

Существует много конфигураций программного пакета Oracle (табл. 4.5). Во-первых, есть две различные версии ядра СУБД Oracle: для индивидуального использования (Personal Oracle) и для организаций (Enterprise Oracle). Кроме того, имеется программа для разработки форм и отчетов (Forms and Reports), программа Oracle Designer и множество средств для публикации баз данных Oracle в WWW.

Таблица 4.5

Семейства программных продуктов Oracle

Программный продукт	Описание
Oracle 9i Database Enterprise Edition	Сервер баз данных для большого количества пользователей или базы данных большого размера, с усовершенствованным управлением, расширяемостью и эффективностью; предназначен для ответственных приложений OLTP (Online Transaction Processing — оперативная обработка транзакций) и хранилищ данных; ориентирован на рынок интернет-приложений и отвечает самым строгим требованиям к качеству обслуживания; обладает возможностями кластеризации, мощными и экономичными средствами безопасности, исключает потери данных и позволяет интерактивно обмениваться информацией
Oracle 9i Database Standard Edition	Сервер баз данных для небольшого числа пользователей и базы данных небольшого размера
Oracle 9i Database Personal	Однопользовательская версия сервера, обычно предназначенная для разработки приложений, функционирующих под управлением Oracle 9i Database Standard/Oracle 9i Database Enterprise

Окончание табл. 4.5

Программный продукт	Описание
Oracle 9i Lite	Упрощенная машина базы данных для систем мобильной связи и небольших офисов, для обработки данных на карманных и портативных компьютерах (ноутбуках)
Oracle 9i Internet Developer Suite (IDS)	Содержит полный набор интегрированных средств быстрого создания интернет-приложений для настраиваемых Web-порталов и развертывания Web-сервисов
Oracle9i Application Server (Oracle9i AS)	Сервер приложений, позволяющий решать широкий спектр задач по поддержке приложений в интернет- и интранет-системах
Oracle Database 10g	Первый в мире сервер баз данных, специально предназначенный для работы в сетях распределенных вычислений (Grid); служит для эффективного развертывания на базе различных типов оборудования, от небольших серверов до мощных многопроцессорных серверных систем, от отдельных кластеров до корпоративных распределенных вычислительных систем; поставляется в одной из четырех редакций в зависимости от масштаба информационной системы, в рамках которой предполагается ее применение: Enterprise Edition, Standard Edition, Standard Edition One, Personal Edition
Oracle Developer Suite 10g	Полный набор интегрированных средств для разработки интернет-приложений; включает удобную интегрированную среду разработки со средствами проектирования баз данных (Oracle Designer) и хранилищ данных (Oracle Warehouse Builder), программирования, разработки компонентов, бизнес-анализа и составления отчетов.

Oracle SQL Plus — это утилита для обработки запросов на языке SQL и создания таких компонентов, как хранимые процедуры и триггеры. Данная утилита неизменно присутствует во всех вариантах конфигурации продукта. С помощью SQL Plus можно передавать Oracle команды на языках SQL и PL/SQL. PL/SQL — это язык, расширяющий возможности SQL за счет включения в него конструкций, характерных для языков программирования.

Помимо этого, есть еще несколько служебных программ Oracle. Так, имеется утилита Oracle Loader для ввода больших объемов информации в базу данных Oracle. Другие утилиты

предназначены для измерения и оптимизации производительности Oracle.

Oracle поддерживает объектно-ориентированные структуры, которые разработчики могут использовать для создания собственных абстрактных типов данных. С помощью Oracle можно также создавать и обрабатывать базы данных, представляющие собой гибриды традиционных и объектных баз данных. Такие гибриды называются *объектно-реляционными базами данных*. Типы данных (табл. 4.6) несколько отличаются от стандартных типов SQL.

Таблица 4.6

Некоторые типы данных СУБД Oracle

Тип данных	Назначение	Размер (*size*)
char(*size*)	Сохраняет символьные данные постоянной длины (размер по умолчанию равен 1)	До 2000 байтов
nchar(*size*)	Такое же, как для типа данных char(size), за исключением максимальной длины, определяемой набором национальных символов базы данных	
varchar2(*size*)	Сохраняет символьные данные переменной длины	До 4000 байтов
nvarchar2(*size*)	Такое же, как для типа данных varchar2, и с теми же условиями, как и для типа данных nchar	
varchar	В настоящее время такое же, как и для varchar2, но рекомендуется использование varchar2, так как в более позднем выпуске varchar может стать отдельным типом данных с отличающейся семантикой сравнения	До 2000 байтов
number(*l*, *d*)	Хранит числа с фиксированной точкой или с плавающей точной, где *l* — длина, а *d* — количество десятичных цифр в дробной части. Например, число типа number (5, 2) не может быть больше, чем 999.99	$1{,}0 \cdot 10^{-130}$... $9{,}99 \cdot 10^{+125}$
decimal(*l*, *d*), dec (*l*, *d*) или numeric(*l*, *d*)	Такое же, как и для number. Предусмотрен для совместимости со стандартом языка SQL	
integer, int или smallint	Предусмотрен для совместимости со стандартом языка SQL. Преобразуется в тип данных number(38)	

Окончание табл. 4.6

Тип данных	Назначение	Размер (*size*)
date	Хранит даты за период с 1 января 4712 г. (до н. э.) до 31 декабря 4712 г. (н. э.)	
blob	Большой двоичный объект	До 4 гигабайт
clob	Большой символьный объект	До 4 гигабайт
raw(*size*)	Двоичные данные, не имеющие заранее определенного формата (например, последовательность графических символов или оцифрованное изображение)	До 2000 байтов

Версия Oracle для предприятий (Oracle Database Enterprise) предусматривает обработку распределенных баз данных, которые хранятся более чем на одном компьютере.

Oracle поддерживает словарь метаданных. Сами метаданные хранятся в таблице DICT. Запросив информацию из этой таблицы, можно ознакомиться с содержимым словаря. Oracle поддерживает три уровня изоляции транзакций: завершенное чтение, сериализуемость и только чтение.

С помощью Oracle администратор определяет пользователей и привилегии. Роль — это группа привилегий и других ролей. Одному и тому же пользователю может быть дано множество ролей, и одна и та же роль может быть дана множеству пользователей.

В процессе резервного копирования и восстановления в Oracle используются три типа файлов: файлы данных, текущие и архивные файлы отката и управляющие файлы. При работе в режиме ARCHIVELOG Oracle записывает в журнал все изменения, произведенные в базе данных. В случае сбоя приложения или экземпляра Oracle может восстановить базу данных без использования архивного файла журнала. Однако для восстановления после сбоя носителя данных архивные файлы необходимы. Резервные копии могут быть согласованными и несогласованными. Несогласованную копию можно сделать согласованной, обработав архивный файл журнала.

4.5. InterBase

SQL-сервер InterBase реализует версию языка SQL, совместимую со стандартом SQL-92 (SQL2), является наиболее

приспособленным для использования в приложениях, разработанных с помощью системы Delphi, и не требует установки дополнительных драйверов. InterBase прост в установке, настройке и администрировании по сравнению с другими SQL-серверами и обладает хорошими функциональными возможностями. Кроме того, следует отметить появление таких бесплатно распространяемых аналогов (клонов) этого сервера, как Firebird и Yaffil [11].

SQL-сервер InterBase обеспечивает хранение и обработку больших объемов информации при одновременной работе с БД множества клиентских приложений. Чтобы максимально разгрузить клиентские приложения от вычислительной работы и гарантировать высокую защищенность и целостность информации, InterBase реализует многие возможности, предусмотренные в языке SQL. Так, для задания ограничений целостности можно определить:

1) связи между таблицами БД путем указания первичных ключей (PRIMARY KEY) у главных таблиц и внешних ключей (FOREIGN KEY) у подчиненных таблиц;

2) ограничения на значения, хранящиеся в отдельных столбцах (CHECK, CONSTRAINT);

3) триггеры (TRIGGER) — подпрограммы, автоматически выполняемые SQL-сервером до и/или после изменения в таблице БД;

4) генераторы (GENERATOR) для создания и использования уникальных значений, хранящихся в определенных столбцах.

Для ускорения взаимодействия клиентских приложений с удаленной БД можно использовать хранимые процедуры (STORED PROCEDURE), представляющие собой подпрограммы, способные посылать запросы к БД и выполнять условные ветвления и циклическую обработку. Хранимые процедуры пишутся с использованием специальных языковых конструкций и содержат часто повторяющиеся последовательности запросов к БД. Текст процедур размещается на сервере в откомпилированном виде. При использовании хранимых процедур:

1) не требуется синтаксическая проверка каждого запроса и его компиляция перед выполнением, что ускоряет выполнение запроса;

2) из клиентского приложения исключается реализация запросов, находящихся в теле хранимых процедур;

3) ускоряется обработка транзакций, т.е. нескольких последовательных логически связанных операторов языка SQL, которые рассматриваются как единое целое, так как вместо длинного SQL-запроса по компьютерной сети передается короткое обращение к хранимой процедуре. Транзакция переводит базу данных из одного целостного состояния в другое.

Таблицы в БД InterBase состоят из содержащих информацию столбцов, типы которых перечислены в табл. 4.7. В частности, столбцы типа BLOB (Binary Large Object — большой двоичный объект) предназначены для хранения больших объемов информации в виде последовательности байтов. Таким образом могут храниться текстовые и графические данные, а также мультимедийная информация. Смысловая интерпретация этой информации выполняется приложением, но разработчик БД может задать так называемые BLOB-фильтры для автоматического преобразования хранящейся информации к другому виду.

Таблица 4.7

Типы данных InterBase

Тип	Размер, байт	Описание
SMALLINT	2	Целочисленные значения от –32768 до +32767
INTEGER	4	Целочисленные значения от –2 147 483 648 до +2 147 483 647
FLOAT	4	Вещественные числа до 7 значащих цифр в диапазоне от $3{,}4\cdot10^{-38}$ до $3{,}4\cdot10^{+38}$
DOUBLE PRECISION	8	Вещественные числа до 15 значащих цифр в диапазоне от $1{,}7\cdot10^{-308}$ до $1{,}7\cdot10^{+308}$
CHAR(*n*) или CHARACTER(*n*)	1—32767	Символьный столбец длиной в *n* символов
VARCHAR(*n*) или CHAR VARING(*n*) или CHARACTER VARING(*n*)	1—32765	Символьный столбец переменной длины, содержащий до *n* символов
DATE	8	Дата в диапазоне от 01.01.0100 до 11.01.5941. Также может содержать сведения о времени
BLOB	Переменный	Двоичные данные любого типа

InterBase позволяет задавать определяемые пользователем функции UDF (User-Defined Function), в которых могут быть реализованы вычисления значений, не предусмотренные в стандартных функциях InterBase. Эти функции разрабатываются на любом языке программирования, позволяющем создавать динамически подключаемые библиотеки DLL (Dynamic Link Library), в частности, на языке Object Pascal.

Физическая модель базы данных, с которой работает InterBase, представляет собой последовательность пронумерованных начиная с нуля страниц. Нулевая страница является служебной и содержит информацию, необходимую для подключения к БД. Размер страницы может быть равен 1 (по умолчанию), 2, 4 или 8 килобайт. Одна страница считывается SQL-сервером за один логический доступ к БД. Некоторые характеристики SQL-сервера InterBase приведены в табл. 4.8.

Таблица 4.8

Некоторые характеристики SQL-сервера InterBase

Характеристика	Значение
Максимальный размер одной БД	Рекомендуется не более 10 гигабайт
Максимальное число таблиц в одной БД	65 536
Максимальное число столбцов (полей) в одной таблице	1000
Максимальное число строк (записей) в одной таблице	Не ограничено
Максимальная длина записи	64 килобайта (не считая полей BLOB)
Максимальная длина поля	32 килобайта (кроме полей BLOB)
Максимальная длина поля BLOB	Не ограничена
Максимальное число индексов в БД	65 536
Максимальное число полей в индексе	16
Максимальное число вложенных SQL- запросов	16
Максимальный размер хранимой процедуры или триггера	48 килобайт

Существует версия SQL-сервера InterBase, реализующая полный набор функций для локального однопользовательского применения. Эта версия, называемая локальным сервером

InterBase, может использоваться при разработке клиент-серверных приложений в качестве модели реального SQL-сервера или для переноса локальной БД на SQL-сервер. Кроме того, локальный сервер InterBase может применяться в качестве процессора БД в обычных локальных приложениях. Его применение позволяет программисту повысить надежность разрабатываемого приложения и избежать возможной потери данных при тестировании неотлаженных приложений.

Если БД, для работы с которой предназначено разрабатываемое приложение, уже существует, то локальный сервер InterBase может быть использован в качестве проверочной модели перед последующим подключением приложения к удаленному SQL-серверу InterBase.

Если реальная БД еще не существует, то локальный сервер InterBase может использоваться для создания прототипа данных, на которых будет проверяться работоспособность приложения.

Если приложение разрабатывается для уже существующей БД, функционирующей на удаленном SQL-сервере InterBase, то перед проверкой работоспособности приложения на реальных данных локальный сервер InterBase может использоваться для создания резервных копий данных или для отладки приложения на представительной выборке информации из существующей БД.

При переносе локальной БД на клиент-серверную платформу локальный сервер InterBase используется в качестве промежуточного сервера, на котором проверяется структура новой БД, предназначенной для установки на сервере. После успешной проверки база данных переносится на SQL-сервер.

Глава 5

ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ НА ОСНОВЕ ПРИНЦИПОВ НОРМАЛИЗАЦИИ

5.1. Цели проектирования реляционных баз данных

Среди множества целей проектирования наиболее важными являются [5]:

1) возможность хранения всех необходимых данных в БД;

2) исключение избыточности данных;

3) сведение числа хранимых в БД отношений к минимуму;

4) нормализация отношений для упрощения решения проблем, связанных с обновлением и удалением данных.

Рассмотрим каждую из целей отдельно.

Возможность хранения всех необходимых данных в БД. БД должна содержать все данные, представляющие интерес для решаемой задачи. Первым шагом в процессе проектирования является определение всех атрибутов, которые впоследствии будут помещены в БД. После определения атрибутов проектировщик решает, сколько отношений необходимо и какие атрибуты включать в какие отношения.

Исключение избыточности данных. Суть этой цели станет понятной, если уяснить четкое различие между дублированием данных и избыточным дублированием данных. Рассмотрим отношение, приведенное на рис. 5.1, *а*. Отношение П-3 имеет два атрибута: НП (номер преподавателя) и ЗавК (фамилия зав. кафедрой). В отношении содержатся данные, указывающие непосредственного начальника каждого преподавателя в институте.

П-З

НП	ЗавК
102	Шаньгин
104	Вернер
108	Вернер
125	Шаньгин

а

П-З

НП	ЗавК
102	Шаньгин
104	Вернер
108	—
125	—

б

Рис. 5.1. **Дублирование данных, не являющееся избыточным**

Фамилии зав. кафедрами могут неоднократно появляться в отношении, что и очевидно из рисунка. Но несмотря на дублирование фамилий, ни одна из повторяемых фамилий не является избыточной. Отсутствие избыточности легко обнаруживается, если из отношения удалить одну из дублируемых фамилий (рис. 5.1, *б*). Очевидно, что в результате удаления теряется информация: невозможно определить фамилии зав. кафедрами, на которых работают преподаватели с номерами 108 и 125.

На рис. 5.2, *а* приведен пример отношения с избыточным дублированием данных. Отношение П-З-Т похоже на отношение П-З, но включает дополнительный атрибут Нтел, представляющий собой номер телефона зав. кафедрой.

П-З-Т

НП	ЗавК	Нтел
102	Шаньгин	2854
104	Вернер	2882
108	Вернер	2882
125	Шаньгин	2854

а

П-З-Т

НП	ЗавК	Нтел
102	Шаньгин	2854
104	Вернер	2882
108	Вернер	—
125	Шаньгин	—

б

П-З

НП	ЗавК
102	Шаньгин
104	Вернер
108	Вернер
125	Шаньгин

З-Т

ЗавК	Нтел
Шаньгин	2854
Вернер	2882

в

Рис. 5.2. **Избыточное дублирование данных и исключение избыточных данных**

В приведенном экземпляре отношения номера телефонов Шаньгина и Вернера появляются более чем один раз, и дублированная информация о телефонных номерах является избыточной. Причина избыточности в том, что, если, например, удалить один из телефонных номеров Шаньгина, то эта информация может быть получена из других кортежей отношения. Из рис. 5.2, *б* видно, что телефонные номера

Шаньгина и Вернера не утеряны, поскольку каждый из них обнаруживается в одном из кортежей отношения.

Такой способ исключения избыточности плох по двум причинам. Во-первых, пустых полей в БД следует избегать, так как при их наличии необходимо дополнительное программирование, направленное на определение действительных значений пустых полей. Во-вторых, что более важно, отношение, представленное на рис. 5.2, *б*, имеет структуру, которая создает серьезные проблемы при удалении информации. Если преподаватель с номером НП=102 уволится с кафедры и кортеж <102, Шаньгин, 2854> будет удален из отношения, произойдет утеря телефонного номера Шаньгина, поскольку нигде в отношении этот номер больше не представлен.

Чтобы исключить избыточность телефонных номеров, отношение П-З-Т заменяется двумя отношениями, одно из которых П-З содержит информацию о номерах преподавателей и фамилиях зав. кафедрами, а другое З-Т — информацию о телефонных номерах зав. кафедрами (рис. 5.2, *в*). Теперь преподаватель с номером 102 может быть удален из отношения П-З без потери номера телефона бывшего начальника этого преподавателя.

Сведение числа хранимых в БД отношений к минимуму. Разбиение одного отношения на два или более меньших отношений желательно с точки зрения исключения определенных проблем, но это неудобно для пользователя БД. Поэтому нельзя допускать неограниченный рост числа отношений.

Нормализация отношений. Для некоторых отношений очень важна проблема удаления и обновления (например, обсуждавшаяся потеря телефонного номера зав. кафедрой). Необходимо уметь обнаруживать эти потенциально опасные отношения и «нормализовать их» посредством разбиения предписанным образом.

5.2. Нормализация

Нормализация — это разбиение отношения на два или более отношений, обладающих лучшими свойствами при включении, обновлении и удалении данных. Окончательная цель нормализации сводится к получению такой базы данных, в которой каждый факт появляется лишь в одном месте, т.е. исключена избыточность информации. Это делается не только

с целью экономии памяти, но и для исключения возможной противоречивости хранимых данных.

Проектирование реляционной базы данных может быть выполнено путем декомпозиции (разбиения), при которой исходное множество отношений, входящих в базу данных, заменяется другим множеством с большим числом отношений, являющихся проекциями исходных отношений.

Процесс проектирования с использованием декомпозиции состоит в последовательной нормализации отношений, и при этом каждое следующее полученное множество отношений соответствует нормальной форме более высокого порядка и обладает лучшими свойствами по сравнению с предыдущим. В теории реляционных баз данных особо выделяются следующие нормальные формы отношений: первая нормальная форма (1НФ), вторая нормальная форма (2НФ), третья нормальная форма (3НФ), нормальная форма Бойса — Кодда (НФБК), четвертая нормальная форма (4НФ), пятая нормальная форма (5НФ).

Универсальным отношением проектируемой базы данных принято называть отношение, содержащее все представляющие интерес атрибуты и имеющее структуру, в которой каждый кортеж состоит из атомарных, т.е. неделимых, и непустых значений атрибутов. Универсальное отношение соответствует *первой нормальной форме*.

С использованием универсального отношения связаны три специфичные проблемы [5]: проблема, обусловленная необходимостью включения новых кортежей в БД; проблема, связанная с обновлением данных в БД; проблема, обусловленная необходимостью удаления кортежей из БД. Эти проблемы называются *аномалиями вставки, обновления и удаления*, поскольку под аномалией понимается отклонение от нормы.

Рассмотрим суть этих проблем на примере БД, содержащей сведения о жителях, их доходах и телефонах. Универсальное отношение Zgrad для этой БД (табл. 5.1) содержит кортежи со значениями атрибутов объектов, показанных на рис. 5.3.

Аномалия вставки. Если появляется новый житель (например, новорожденный), у которого отсутствуют источники дохода, то для него приходится включать в отношение Zgrad строку с пустыми значениями полей Source и Money. Но поскольку пустых (т.е. неопределенных) полей в отношении (таблице) быть не должно, то регистрация нового жителя в БД откладывается до появления у него источника дохода.

Аномалия обновления. Отношение Zgrad характеризуется как явной, так и неявной избыточностью. *Явная избыточность* заключается в том, что фамилия жителя, его дата рождения, пол, адрес, номер телефона и значения других атрибутов могут появиться в отношении несколько раз. Например, у Петровой П. П. адрес появляется в отношении дважды (см. табл. 5.1). Если она сообщит об изменении своего адреса, то придется проследить изменение ее адреса в обоих кортежах во избежание противоречивости данных.

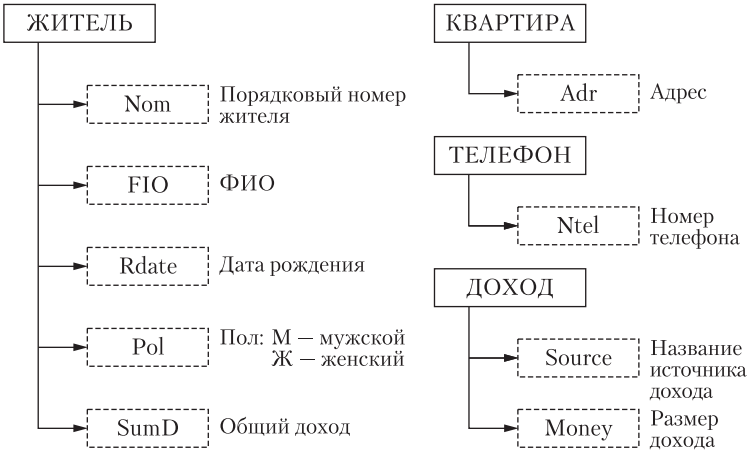


Рис. 5.3. **Объекты предметной области и их атрибуты**

Таблица 5.1

Универсальное отношение (однотабличная база данных Zgrad)

Nom	FIO	Rdate	Pol	Adr	Ntel	Source	Money	SumD
1	Иванов И. И.	11.03.78	м	801-12	531-5894	Работа1	1500	3500
1	Иванов И. И.	11.03.78	м	801-12	531-5894	Работа2	2000	3500
2	Петрова П. П.	01.12.79	ж	05-321	Нет	Стипендия	1000	2500
2	Петрова П. П.	01.12.79	ж	05-321	Нет	Работа1	1500	2500
3	Иванова И. И.	14.04.30	ж	801-12	531-5894	Пособие	500	2500
3	Иванова И. И.	14.04.30	ж	801-12	531-5894	Пенсия	2000	2500

Окончание табл. 5.1

Nom	FIO	Rdate	Pol	Adr	Ntel	Source	Money	SumD
4	Ильин Ф. П.	25.06.70	м	901-323	534-9900	Работа3	3000	3000
5	Иванов И. И.	23.10.28	м	801-12	531-5894	Пенсия	3000	3000

В отношении Zgrad *неявная избыточность* обнаруживается в том, что один и тот же номер телефона имеют все жители, живущие в одной квартире. Например, номер телефона для квартиры с адресом 801-12 появляется в сочетании с фамилиями трех жителей. Допустим, Иванов И. И. 1978 года рождения известит служащего, работающего с базой данных, о том, что его номер телефона изменен на 531-55-33, забыв при этом сообщить о других жильцах квартиры. Если служащий изменит номер телефона только в тех строках, которые содержат телефонный номер Иванова И. И. 1978 года рождения, то правильный номер телефона в квартире с адресом 801-12 будет фактически утерян, поскольку в отношении Zgrad будут присутствовать два различных телефонных номера для одной квартиры.

Аномалия удаления. В отношении Zgrad присутствует только один кортеж, соответствующий Ильину Ф. П. Предположим, что работающий с базой данных служащий узнает, что этот житель лишился своего источника дохода, условно именуемого Работа3, и удаляет этот кортеж из отношения. Поскольку это единственный кортеж с информацией об Ильине, его удаление приведет к исключению жителя из БД. Если служащий вслед за этим запросит список жителей, зарегистрированных в БД, то Ильина в списке не окажется, хотя он продолжает жить по прежнему адресу.

5.3. Функциональные зависимости

Процесс разбиения отношения с целью уменьшения вероятности возникновения аномалий называется *декомпозицией*. Для разбиения исходного отношения на ряд новых отношений, называемых проекциями отношения, используется соответствующая операция реляционной алгебры. *Полная декомпозиция отношения* — это совокупность произвольного числа его проекций, естественное соединение которых идентично этому отношению. Например, проекции *R1* и *R2,*

показанные на рис. 2.8, образуют полную декомпозицию исходного отношения *R*, показанного на рис. 2.7.

Ключевой для осуществления декомпозиции является концепция функциональных зависимостей между атрибутами в рассматриваемом отношении [5]. *Функциональная зависимость (ФЗ)* определяется следующим образом: если даны два атрибута *A* и *B*, то говорят, что *B* функционально зависит от *A*, если для каждого значения *A* в любой момент времени существует ровно одно связанное с ним значение *B*. *A* и *B* могут быть составными, т.е. представлять собой не единичные атрибуты, а группы, состоящие из двух и более атрибутов.

Смысл данного определения состоит в том, что если *B* функционально зависит от *A*, то каждый из кортежей, имеющих одно и то же значение *A*, должен иметь также одно и то же значение *B*. Значения *A* и *B* могут изменяться время от времени, но при этом они должны изменяться так, чтобы каждое уникальное значение *A* имело только одно значение *B*, связанное с ним.

Два наиболее часто используемых способа нотации для описания ФЗ показаны на рис. 5.4.

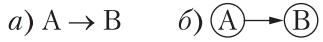


Рис. 5.4. **Способы описания ФЗ:**
а — математический; *б* — графический

В конкретной ситуации ФЗ определяется путем анализа смысловых (семантических) свойств всех атрибутов в отношении и посредством вывода заключения о том, как атрибуты соотносятся между собой, т.е. ФЗ необходимо получить исходя из базовых свойств самих атрибутов.

В качестве примера обратимся к атрибутам в универсальном отношении Zgrad (см. табл. 5.1) и посмотрим, как эти атрибуты связаны между собой. После изучения семантики атрибутов выявлены существующие зависимости, приведенные на рис. 5.5.

Соображения, объясняющие наличие этих ФЗ, таковы:

1) номера жителей являются уникальными. Каждому жителю назначается номер Nom, причем все номера различны. Таким образом, если известен номер жителя, то с ним может быть связана только одна фамилия, имя и отчество FIO: *Nom→FIO*. Обратное не является верным: FIO→Nom

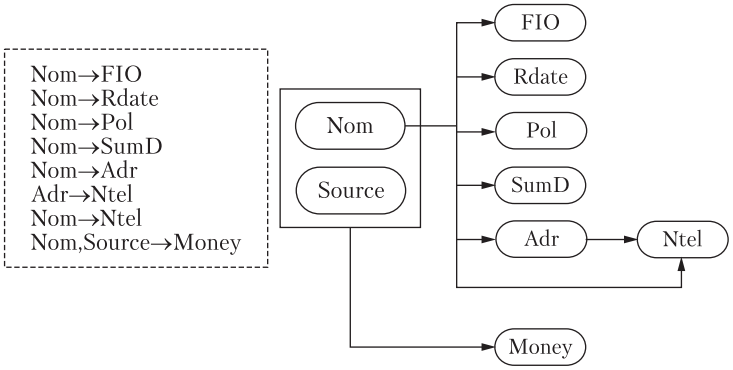


Рис. 5.5. **Различные способы представления ФЗ, существующих между атрибутами отношения Zgrad**

не является правильной ФЗ, поскольку несколько жителей могут иметь одинаковые фамилию, имя и отчество;

2) аналогичные рассуждения относятся и атрибутам Rdate, Pol и SumD. Следовательно, если известен номер жителя, то с ним может быть связана только одна дата рождения, один пол и один общий доход: *Nom→Rdate*, *Nom→Pol*, *Nom→SumD*. Обратное не является верным: Rdate→Nom, Pole→Nom, SumD→Nom не являются правильными ФЗ, поскольку несколько жителей могут иметь одинаковые даты рождения, пол и общий доход;

3) каждый житель зарегистрирован в одной квартире, но в одной квартире может проживать более чем один житель. Следовательно, *Nom→Adr* является верной ФЗ, а Adr→Nom — нет;

4) поскольку для каждой квартиры будет учитываться не более одного телефона, то по адресу квартиры однозначно определяется номер телефона или его отсутствие: *Adr→Ntel*. Обратное, т.е. Ntel→Adr, не является верным, поскольку не во всех квартирах есть телефоны, и при отсутствии телефона адрес квартиры не определить;

5) зная номер жителя, можно определить номер телефона или его отсутствие в квартире, занимаемой жителем: *Nom→Ntel*;

6) размер источника дохода Money, имеющегося у жителя, однозначно определяется только в том случае, если известен номер жителя Nom, имеющего источник дохода Source: *Nom,Source→Money*. Эта ФЗ представляет собой пример сложного набора атрибутов (Nom,Source), входящего в ФЗ.

5.4. Нормальные формы отношений

Каждой нормальной форме соответствует определенный набор условий, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору условий [10]. Так, согласно определению, данному в разделе 5.2, для отношения, находящегося в 1НФ, все его атрибуты имеют неделимые и непустые значения.

Для всех нормальных форм характерно, что каждая следующая нормальная форма определенным образом улучшает свойства предыдущей, а при переходе к следующей нормальной форме свойства предыдущих нормальных форм сохраняются.

Для формулировки условий, которым удовлетворяют нормальные формы, следующие за 1НФ, используется ряд понятий, рассматриваемых далее.

Пусть X и Y — множества, которые содержат один или несколько атрибутов одного отношения. Если X состоит из нескольких атрибутов, то говорят, что Y находится в *полной функциональной зависимости* от X тогда и только тогда, когда (а) Y функционально зависит от X ($X \rightarrow Y$) и (б) Y функционально не зависит от любого X', где X' — такое подмножество атрибутов множества X, что по меньшей мере один атрибут их X не принадлежит X', т.е. $X' \subset X$.

Функциональная зависимость $A \rightarrow C$ называется *транзитивной*, если существуют зависимости $A \rightarrow B$ и $B \rightarrow C$. Например, для отношения Zgrad транзитивной является зависимость Nom→Ntel из-за наличия зависимостей Nom→Adr и Adr→Ntel.

Возможный ключ отношения представляет собой атрибут или набор атрибутов, который может быть использован для данного отношения в качестве первичного ключа. Первичный ключ всегда является возможным ключом; однако не исключено наличие других возможных ключей, которые могли бы быть, но не были использованы в качестве первичного ключа.

Неключевым атрибутом называется любой атрибут отношения, не входящий в состав ни одного возможного ключа отношения.

Взаимно-независимые атрибуты — это атрибуты отношения, не зависящие функционально друг от друга.

Если в отношении имеется несколько ФЗ, то каждый атрибут или набор атрибутов A, от которого в полной функцио-

нальной зависимости находится другой атрибут *B*, называется *детерминантом отношения*.

Функциональная зависимость, не заключающая в себе такой информации, которая не могла бы быть получена на основе других зависимостей из числа использованных при проектировании базы данных, называется *избыточной ФЗ*. Поскольку избыточная ФЗ не содержит уникальной информации, она может быть удалена из набора ФЗ без отрицательного воздействия на результаты.

Отношение находится во *второй нормальной форме (2НФ)* тогда и только тогда, когда отношение находится в 1НФ и *все* его атрибуты, не входящие в первичный ключ, связаны полной функциональной зависимостью с первичным ключом. Отношение Zgrad не находится в 2НФ, поскольку его первичным ключом является пара атрибутов <Nom,Source> и, в частности, атрибут SumD, не входящий в первичный ключ, не связан полной функциональной зависимостью с первичным ключом из-за наличия зависимости Nom→SumD.

Отношение находится в *третьей нормальной форме (3НФ)* только в том случае, когда (1) оно находится в 2НФ и (2) ни один из его неключевых атрибутов не зависит функционально от любого другого неключевого атрибута. Второе условие эквивалентно требованию отсутствия транзитивных зависимостей в отношении.

Одним из самых первых и самых важных результатов в теории реляционных БД стало доказанное Коддом утверждение о том, что большинство потенциальных аномалий в БД будет устранено в случае должной декомпозиции каждого отношения в нормальную форму Бойса — Кодда [5]. Отношение находится в *нормальной форме Бойса — Кодда (НФБК)* тогда и только тогда, когда (1) оно находится в 3НФ и (2) каждый детерминант отношения является возможным ключом отношения. Например, для отношения Zgrad второе из указанных условий не выполняется, так как не каждый детерминант отношения является возможным ключом:

Возможные ключи	Детерминанты
<Nom, Source>	<Nom, Source> <Nom> <Adr>

Отношение находится в *четвертой нормальной форме (4НФ)* только в том случае, если оно находится в НФБК и ка-

ждая его полная декомпозиция из двух его проекций такова, что обе проекции не содержат общего возможного ключа.

Отношение находится в *пятой нормальной форме (5НФ)* тогда и только тогда, когда оно находится в 4НФ и в каждой его полной декомпозиции все проекции содержат возможный ключ.

Хотя существуют 4НФ и 5НФ, которые по сравнению с НФБК накладывают более сильные ограничения на разрабатываемые отношения, на практике большинство проектировщиков стараются получить отношения в НФБК.

5.5. Общий подход к декомпозиции отношений

Проектирование реляционной БД методом декомпозиции отношений в простейшем случае состоит из четырех шагов [5].

1. Разработка универсального отношения для БД.
2. Определение всех ФЗ между атрибутами отношения.
3. Определение того, находится ли отношение в НФБК. Если да, проектирование завершается; если нет, отношение должно быть разбито на два отношения.
4. Повторение шагов 2 и 3 для каждого нового отношения, полученного в результате декомпозиции. Проектирование завершается, когда все отношения будут находиться в НФБК.

Разбиение отношения на два новых отношения осуществляется с помощью ФЗ следующим образом.

Пусть отношение $R(A,B,C,D,E,...)$ не находится в НФБК. Определяется ФЗ, например $C \rightarrow D$, про которую известно, что она служит причиной того, что отношение R не находится в НФБК (т.е. C является детерминатом, но не является возможным ключом). Создаются два новых отношения — проекции отношения R — $R1(A,B,C,E,...)$ и $R2(C,D)$, где зависимая часть ФЗ (т.е. атрибут D) была выделена из R и опущена при формировании отношения *R1*, а ФЗ была использована полностью при формировании отношения *R2*. После этого нужно проверить, находятся ли в НФБК отношения *R1* и *R2*.

Этот тип декомпозиции называется декомпозицией без потерь при естественном соединении.

В качестве примера использования описанного метода проведем декомпозицию отношения Zgrad. Обращаясь к определенным в предыдущем разделе детерминантам и воз-

можным ключам этого отношения, видим, что имеются два детерминанта, которые не являются возможными ключами:

<Nom> и <Adr>.

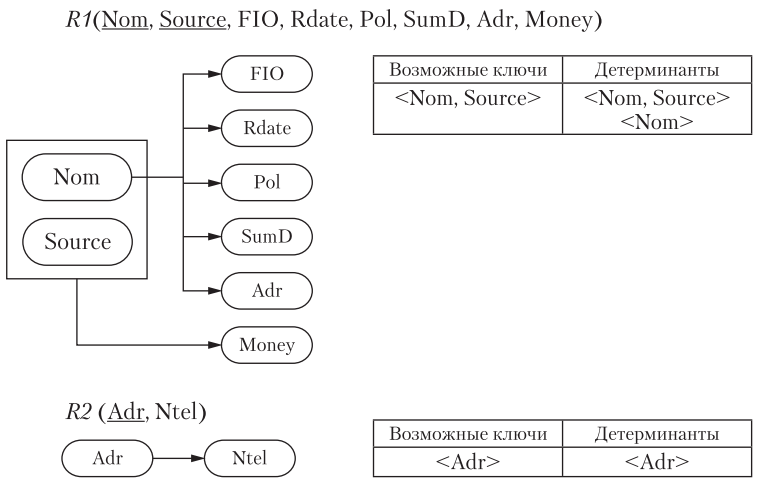
Перед началом декомпозиции разрабатывается универсальное отношение

Zgrad (Nom, Source, FIO, Rdate, Pol, Adr, Ntel, Money, SumD) и определяются все ФЗ между атрибутами отношения (см. рис. 5.5).

Кандидатами среди ФЗ для осуществления проекций являются ФЗ, зависящие от детерминантов Nom и Adr: Nom→FIO, Nom→Rdate, Nom→Pol, Nom→SumD, Nom→Adr, Adr→Ntel, Nom→Ntel.

Далее нужно решить, какую ФЗ следует выбрать для проведения первой проекции. Простым правилом выбора ФЗ для проекции может служить поиск «цепочки ФЗ» вида $A \rightarrow B \rightarrow C$ с последующим использованием для проекции крайней правой зависимости. В нашем случае такой «цепочкой» является Nom→Adr→Ntel и «конец цепочки» Adr→Ntel порождает первую проекцию. Полученные в итоге отношения *R1* и *R2* приведены на рис. 5.6 вместе с их ФЗ.

Поскольку все детерминанты отношения *R2*(Adr, Ntel) являются возможными ключами, то оно находится в НФБК и не нуждается более в декомпозиции. Однако отношение *R1*(Nom,



Возможные ключи	Детерминанты
<Nom, Source>	<Nom, Source> <Nom>

Возможные ключи	Детерминанты
<Adr>	<Adr>

Рис. 5.6. **Отношения *R1* и *R2* как результат проекций отношения Zgrad**

Source, FIO, Rdate, Pol, SumD, Adr, Money) не находится в НФБК, так как детерминант <Nom> не является возможным ключом. Следовательно отношение *R1* необходимо подвергнуть дальнейшему разбиению. Детерминант <Nom>, из-за которого возникло затруднение, имеет пять зависимых от него атрибутов

Nom→FIO, Nom→Rdate, Nom→Pol, Nom→SumD, Nom→Adr,

что можно рассматривать в качестве единой ФЗ с составной правой частью

Nom→FIO,Rdate,Pol,SumD,Adr.

Проекции отношения *R1* на множества атрибутов {Nom, FIO, Rdate, Pol, SumD, Adr} и {Nom, Source, Money} приводят к получению отношений *R3* и *R4*, показанных на рис. 5.7. Эти два отношения находятся в НФБК и вместе с отношением *R2* могут использоваться в качестве даталогической модели БД со сведениями о жителях. На рис. 5.8 представлен окончательный вид отношений для спроектированной БД ZgradDB, а также экземпляры каждого отношения с данными, совпадающими с использованными для исходного отношения Zgrad (см. табл. 5.1).

Из полученной БД видно, что в процессе декомпозиции автоматически произошло разбиение исходного отношения Zgrad на три логических компонента: *R2*, в котором содержится ин-

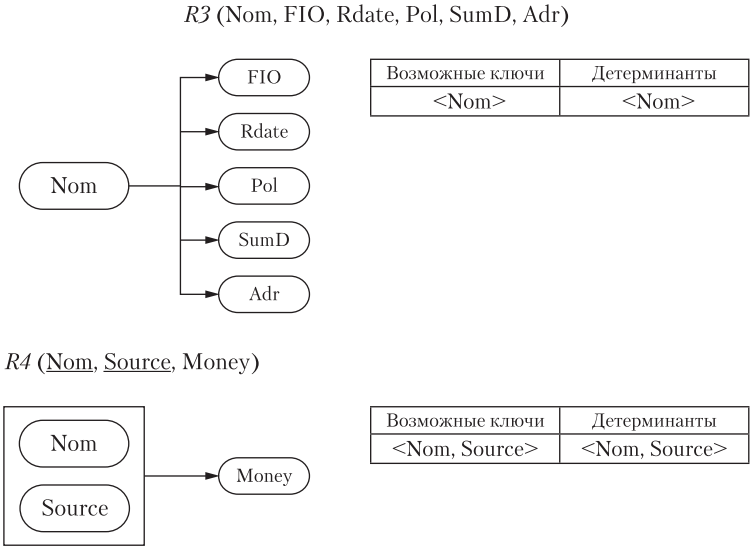


Рис. 5.7. **Отношения *R3* и *R4* как результат проекций отношения *R1***

формация о квартирах и телефонах; *R3*, в котором содержится информация о жителях, их общих доходах и адресах; *R4*, в котором содержится информация об источниках и размерах доходов жителей. Такое логическое разбиение является прямым результатом использования в процессе декомпозиции заложенной в ФЗ информации, детализирующей то, как различные атрибуты в исходном отношении соотносятся друг с другом.

Подобно тому, как проверялась на наличие аномалий вставки, обновления и удаления однотабличная БД Zgrad, следует проверить и спроектированную БД ZgradDB.

Аномалия вставки. Если появляется новый житель, у которого отсутствуют источники дохода, то для него в отношение *R3* включается строка с известными (непустыми) значениями всех полей. В другие отношения строки не вставляются, а факт отсутствия источников доходов у жителя представляется в БД ZgradDB нулевым значением общего дохода SumD. Поскольку пустых (т.е. неопределенных) полей в БД не появляется, то регистрация нового жителя в БД происходит успешно, а следовательно, *аномалия вставки не проявляется*.

R3 (Nom, FIO, Rdate, Pol, SumD, Adr)
R2 (Adr, Ntel)
R4 (Nom, Source, Money)

а

Nom	FIO	Rdate	Pol	SumD	Adr
1	Иванов И. И.	11.03.78	м	3500	801-12
2	Петрова П. П.	01.12.79	ж	2500	05-321
3	Иванова И. И.	14.04.30	ж	2500	801-12
4	Ильин Ф. П.	25.06.70	м	3000	901-323
5	Иванов И. И.	23.10.28	м	3000	801-12

Adr	Ntel
801-12	531-5894
05-321	Нет
901-323	534-9900

Nom	Source	Money
1	Работа1	1500
1	Работа2	2000
2	Стипендия	1000
2	Работа1	1500
3	Пособие	500
3	Пенсия	2000
4	Работа3	3000
5	Пенсия	3000

б

Рис. 5.8. **Спроектированная база данных ZgradDB:**
а — отношения; *б* — экземпляры отношений

Аномалия обновления. В БД ZgradDB существенно уменьшена *избыточность*. Так, независимо от числа доходов у жителя сведения о каждом жителе в отношении *R3* представлены одной строкой. Например, если Петрова П. П. сообщит об изменении своего адреса, то придется изменить ее адрес в единственной строке отношения *R3*, а не во многих строках, как это происходит в БД Zgrad.

В спроектированной БД также исключена избыточность номеров телефонов, установленных в квартирах. Допустим, Иванов И. И. 1978 года рождения известит служащего, работающего с базой данных, о том, что его номер телефона изменен на 531-55-33. Служащий изменит номер телефона только в одной строке отношения *R2*, содержащей телефонный номер, соответствующий квартире с адресом 802-12, в которой проживает Иванов И. И. 1978 года рождения. При этом для всех жителей этой квартиры в БД окажется сохраненным правильный номер телефона.

Рассмотренные примеры обновления данных свидетельствуют, что в БД ZgradDB *аномалия обновления не проявляется.*

Аномалия удаления. Предположим, что работающий с базой данных служащий узнает, что житель Ильин Ф. П. лишился своего источника дохода, условно именуемого Работа3, и удаляет из отношения *R4* кортеж <4, Работа3, 3000>. Вся другая информация об Ильине в БД сохраняется. Если служащий вслед за этим запросит список жителей, зарегистрированных в БД, в нем окажутся сведения и об Ильине, т.е. в БД ZgradDB *аномалия удаления не проявляется.*

Следовательно, НФБК действительно гарантирует устранение большинства аномалий в базах данных.

5.6. Анализ полученного набора отношений

Полученный набор отношений, находящихся в НФБК и рассматриваемых в качестве окончательной ДЛМ БД, необходимо проверить на предмет наличия невыявленных проблем.

1. Составляются списки ФЗ для каждого отношения. Эти списки проверяются на соответствие двум условиям: во-первых, каждая ФЗ не должна появляться более чем в одном отношении; и во-вторых, набор ФЗ, полученный в результате проектирования, должен в точности совпадать с набором, присутствующим в *минимальном покрытии*, полученном из

исходного набора ФЗ путем удаления избыточных ФЗ перед началом декомпозиции. В противном случае необходимо показать возможность получения итогового набора ФЗ из минимального покрытия с помощью специальных правил вывода. Если хотя бы одно из этих условий не соблюдается, следует проанализировать процесс проектирования для выявления ошибок и рассмотреть другие варианты проектирования.

2. Осуществляется проверка на присутствие *избыточных отношений*. Отношение является избыточным, если (а) все атрибуты в избыточном отношении могут быть найдены в другом отношении ДЛМ; (б) все атрибуты в избыточном отношении могут быть найдены в отношении, которое получается из других отношений ДЛМ с помощью серии операций соединения над этими отношениями. Если устанавливается избыточность отношения, его следует исключить из ДЛМ.

Для примера, иллюстрирующего первый тип избыточности, возьмем ДЛМ, состоящую их трех отношений:

R1(*A*, *B*)

R2(*B*, *C*, *Y*, *Z*)

R3(*A*, *B*, *K*)

Отношение *R1* является избыточным, так как все его атрибуты присутствуют в отношении *R3*.

Для иллюстрации избыточности второго типа возьмем ДЛМ следующего вида:

R4(*A*, *C*, *X*)

R5(*D*, *K*, *F*)

R6(*D*, *E*, *G*, *H*)

R7(*A*, *B*, *D*)

R8(*A*, *B*, *E*, *G*)

Отношение *R8* является избыточным, так как применение операции соединения к *R6* и *R7* (по общему атрибуту *D*) дает в результате отношение *R9*(*A*, *B*, *D*, *E*, *G*, *H*), которое содержит все атрибуты, присутствующие в *R8*.

3. Рассмотрение отношений с практической точки зрения. Изучается характер использования отношений в проектируемой реляционной БД и определяется, будут ли они поддер-

живать те типы запросов и операций обновления, которые предполагается использовать. При анализе следует учитывать информационные потребности пользователей.

Аналогичным проверкам должен быть подвергнут и набор отношений, полученных другими методами (например, рассматриваемым в следующей главе методом, основанном на использовании инфологической модели предметной области). Для этого определяются межатрибутные ФЗ для каждого отношения и проверяется соответствие отношений НФБК.

Глава 6

КОНЦЕПТУАЛЬНОЕ И ДАТАЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

6.1. Необходимость концептуального проектирования

Концептуальное проектирование является основой всего процесса проектирования баз данных. Для того чтобы БД адекватно отражала предметную область, проектировщик БД должен хорошо представлять себе все особенности, присущие данной предметной области, и уметь отобразить их в БД. Поэтому перед началом проектирования БД необходимо детально разобраться, как функционирует предметная область, для отображения которой создается БД. На этом этапе важным является участие заказчика, например, менеджера или экономиста.

Предметная область должна быть предварительно описана. Для этого можно использовать естественный язык, но его применение имеет много недостатков, основными из которых являются громоздкость описания и неоднозначность его трактовки. Поэтому обычно для этих целей используют искусственные формализованные языковые средства.

Целью концептуального проектирования является создание *инфологической модели предметной области* (ИЛМ ПО), которая представляет собой описание ПО, выполненное с помощью специальных языковых средств, не зависящих от используемых в дальнейшем программных средств.

Желательно, чтобы язык спецификации ИЛМ был одинаково применим как при ручном, так и при автоматизированном проектировании. Для этого он должен:

1) быть вычисляемым, т.е. восприниматься и обрабатываться компьютером;

2) использовать удобный пользователю интерфейс, в частности графический интерфейс;

3) быть независимым от оборудования и других ресурсов, подверженных частым изменениям;

4) использовать средства тестирования ИЛМ, а также иметь аппарат для генерации структуры БД после завершения спецификации ИЛМ.

ИЛМ должна легко восприниматься разными категориями специалистов, участвующих в создании информационной системы. ИЛМ является средством коммуникации различных коллективов как конечных пользователей, так и разработчиков. ИЛМ должна содержать необходимую и достаточную информацию для проектирования информационной системы.

ИЛМ включает в себя ряд компонентов (рис. 6.1).

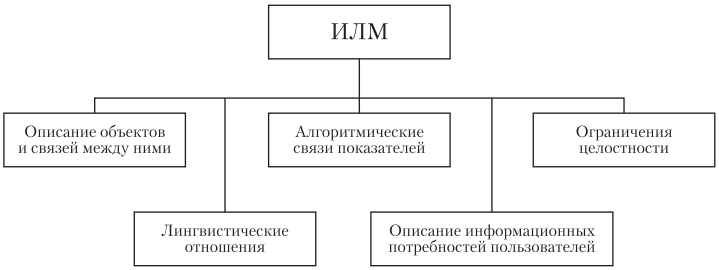


Рис. 6.1. **Компоненты ИЛМ**

Основным компонентом является описание объектов предметной области и связей между ними.

Описание предметной области всегда представляется с помощью какой-либо знаковой системы. Поэтому в ИЛМ отражаются не только отношения, присущие предметной области, но и лингвистические отношения, обусловленные особенностями отображения ПО в языковой среде. Поэтому нужно учитывать такие лингвистические категории, как «синонимия», «омонимия», «изоморфизм» и др.

В ИЛМ должны быть отражены и алгоритмические зависимости между показателями. Для этих целей можно использовать графы взаимосвязи показателей, отражающие, какие показатели служат исходными для вычисления других (рис. 6.2). Расчетные формулы, а также алгоритмы вычислений также должны быть представлены в ИЛМ.

Следующим компонентом является описание информационных потребностей пользователей. Это описание должно отражать тип запроса, режим использования данных и т.п.

Важной характеристикой ПО являются так называемые ограничения целостности, которые также отражаются в ИЛМ. Ограничения целостности — это условия, при которых имеют смысл значения, хранящиеся в БД, или условия, при которых значения могут оказаться записанными в БД.

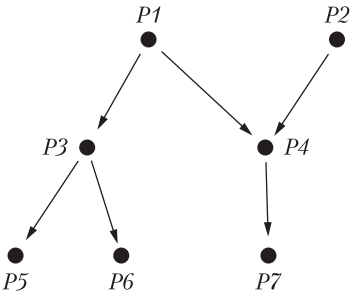


Рис. 6.2. **Граф взаимосвязи показателей**

Проблема целостности данных состоит в обеспечении правильности данных в БД в любой момент времени. Целостность данных обеспечивается набором специальных условий или утверждений, называемых ограничениями целостности. Ограничения целостности — это утверждение о допустимости значений отдельных информационных единиц и связей между ними. Ограничения целостности определяются в большинстве случаев особенностями ПО, хотя могут отражать и чисто информационные характеристики.

Ограничения целостности могут относиться к разным информационным единицам: атрибутам (полям), кортежам (строкам, записям), отношениям (таблицам, файлам), связям между отношениями и т.п.

Для полей чаще всего используются следующие виды ограничений.

1. Тип и формат поля.
2. Задание диапазона значений. Значения диапазона и его тип зависят от особенностей ПО.
3. Признак непустого поля. Характеризует недопустимость пустого значения поля в БД. Например, в отношении, содержащем сведения о сотрудниках, поля «фамилия», «имя»,

«отчество», «оклад» должны обязательно иметь какое-то значение, а у поля «ученая степень» значение может отсутствовать.

4. Задание домена. Поле может принимать значение из заданного множества значений. Например, значением поля «пол» может быть либо «мужской», либо «женский». Значением поля «должность» для профессорско-преподавательского состава может быть одно из следующих значений: «ассистент», «старший преподаватель», «доцент», «профессор». Домен необязательно должен определяться перечислением входящих в него значений.

Как всякая классификация, приведенная выше классификация видов ограничений является условной. Кроме того, домен может определяться и алгоритмически. Например, многие СУБД поддерживают тип поля «ДАТА» и при вводе значений обеспечивают автоматическую проверку на допустимость введенной даты. Поэтому для поддержания целостности данных важно знать о возможностях СУБД и правильно выбрать тип поля.

Специфическим ограничением на значение поля является признак его уникальности. Это ограничение проверяет допустимость значения данного поля, но при этом просматривается вся таблица (файл).

Признак уникальности значения тесно связан с понятием ключа, но уже первичного, так как ключ может быть представлен не только одним полем, а быть составным. Уникальное поле является возможным ключом данного отношения. При наличии нескольких возможных ключей один из них должен быть выбран в качестве первичного ключа. Это поле не должно иметь пустое значение. Не все СУБД поддерживают концепцию ключа и требуют определять его при описании БД.

Рассматриваемое ограничение чаще всего возникает при отображении в БД каких-то объектов, и уникальное поле является идентификатором объекта. Поэтому оно часто называется ограничением целостности объекта.

Рассмотренные выше ограничения определяли проверку значения поля вне зависимости от того, вводится ли это значение впервые или корректируются имеющиеся в БД значения. Ограничения, которые используются только при проверке допустимости корректировки, называются ограничениями перехода. Например, если в БД имеется поле «возраст сотрудника», то при корректировке значение этого поля может только увеличиваться. Если в БД хранится поле

«год рождения», то следует запретить возможность корректировать это поле.

Когда речь идет об ограничениях целостности, относящихся к кортежу, то имеется в виду либо ограничение на значение всей строки, рассматриваемой как единое целое, либо ограничения на соотношения значений отдельных полей в пределах одной строки. Так, естественным ограничением является требование уникальности каждой строки таблицы. По определению в отношении не может быть одинаковых кортежей, но не все реляционные СУБД обеспечивают соблюдение этого ограничения.

В качестве примера ограничений, относящихся ко всей таблице, можно привести следующий. Предположим, что фонд заработной платы формируется исходя из величины средней зарплаты одного сотрудника и эта величина равна *N* руб. Тогда в качестве ограничения целостности таблицы может быть задано условие, указывающее, что среднее значение поля «оклад» должно быть не больше *N*. Примерами ограничения целостности, которыми проверяют соотношения между строками одной таблицы, являются следующие: 1) нельзя быть родителем и ребенком одного и того же человека; 2) год рождения родителя должен быть меньше, чем год рождения ребенка. Первый из приведенных примеров является частным случаем более общего ограничения на отсутствие циклов.

К аналогичным ограничениям относятся ограничения на отсутствие циклов при определении состава изделия (узел не может входить сам в себя), при описании организационной структуры и во многих других случаях. Если СУБД не позволяет контролировать подобные ограничения целостности, то следует написать процедуру, позволяющую делать это.

Все ограничения, которые были рассмотрены ранее, затрагивали информационные единицы в пределах одной таблицы. Кроме такого рода ограничений часто используются ограничения, затрагивающие несколько взаимосвязанных таблиц. Наиболее часто встречающееся из этих ограничений — ограничение целостности связи. Оно выражается в том, что значение атрибута, отражающего связи между объектами и являющегося внешним (вторичным) ключом отношения, обязательно должно совпадать с одним из значений атрибута, являющегося первичным ключом отношения, описывающего соответствующий объект.

Разновидностью ограничения целостности связи является ограничение связи по существованию, заключающееся в том,

что для существования объекта в отношении *R1* необходимо, чтобы он был связан с объектом в отношении *R2*. Например, при приеме на работу каждый из работающих должен быть зачислен в какой-то отдел, и соответствующая запись в таблице «Кадры» в поле «отдел» должна иметь значение, совпадающее с одним из значений соответствующего поля в таблице «Отделы».

Кроме того, ограничения, отражающие связь таблиц могут представлять собой условия, проверяющие отсутствие логических противоречий между данными во взаимосвязанных таблицах. Например, если для каждой должности установлена определенная «вилка» оклада, то значение поля «оклад» в таблице «Кадры» не должно выходить за пределы «вилки», которая зафиксирована в таблице «Должности».

Своеобразным видом ограничений целостности является запрет на обновление. Он может быть обусловлен технологией обработки данных или спецификой ПО. Так, если описывается объект «Личность», то такие атрибуты, как дата рождения и место рождения, являются постоянными (статическими) и меняться не могут.

По моменту контроля за соблюдением ограничений целостности различают безотлагательные (одномоментные) и отложенные ограничения целостности. Отложенные ограничения целостности могут не соблюдаться в процессе выполнения какой-то группы операций, но обязаны быть соблюдены по завершении выполнения этой группы операций. С понятием отложенного ограничения целостности тесно связано понятие транзакции.

Очень важным видом ограничений целостности являются функциональные зависимости. Информация об имеющихся в данной ПО функциональных зависимостях фиксируется в ИЛМ и используется при проектировании БД и для контроля целостности при функционировании БД. Для соответствующих полей в БД желательно задать запрет на обновление.

Запрет на обновление может относиться не только к отдельному полю, но и ко всей строке (записи) и к таблице.

Рассмотрим пример ограничения на обновление строки (записи). Пусть в БД по кадровому составу для каждого из сотрудников хранятся сведения о поощрениях. Эта информация хранится в таблице «Поощрения», имеющей такие атрибуты (поля): табельный номер сотрудника, вид поощрения, дата. В эту таблицу могут добавляться строки, но каждая отдельная запись изменяться не может.

В этом примере наблюдается также ограничение связи по существованию между таблицами «Поощрения» и «Сотрудники»: табельный номер в таблице «Поощрения» должен обязательно присутствовать в таблице «Сотрудники»; при удалении строки из таблицы «Сотрудники» все связанные с ней строки в таблице «Поощрения» должны быть также удалены.

Некоторые СУБД позволяют при описании данных задавать так называемое обязательное членство для включения и каскадное удаление. В этом случае целостность при корректировке будет обеспечиваться системой автоматически и гарантируется ограничение связи по существованию.

Для автоматического контроля целостности эта информация должна быть зафиксирована в словаре данных. Для контроля целостности при выполнении операций реляционной алгебры по меньшей мере должна быть зафиксирована информация о ключах и возможных ключах отношений.

По способу задания ограничения целостности могут быть явными и неявными. Неявные ограничения целостности определяются спецификой модели данных и проверяются СУБД автоматически.

Рассмотренные выше примеры ограничений целостности относились к данным пользователя. Понятие целостности может относиться и к служебной информации. Это прежде всего относится к поддержанию соответствия между индексными файлами и соответствующими им индексируемыми файлами БД.

Наряду с понятием целостности БД может быть введено понятие информационной целостности банка данных, заключающееся в обеспечении правильности взаимосвязи всех его информационных компонентов (файлов БД, программных файлов, описаний форм ввода-вывода, отчетов). Например, если для файла БД имеется связанная с ним форма отчета, то при удалении из файла поля, вывод которого предусмотрен в этой форме, возникает ошибка при выводе отчета. Нарушения целостности могут возникнуть, если изменяется тип данных, хранящихся в поле, и во многих других случаях.

Некоторые СУБД имеют специальный механизм, позволяющий отслеживать согласованность различных информационных компонентов банка данных. Для отслеживания взаимосвязи между всеми информационными компонентами БнД должны использоваться словари данных. Задание ограничений целостности и их проверка являются важной частью проектирования и функционирования БнД.

Ограничения целостности, присущие той или иной ПО, должны быть выявлены при обследовании и зафиксированы в ИЛМ. Вопрос о необходимости проверки ОЦ при функционировании БнД должен решаться на основе анализа эффективности проекта, так как в некоторых случаях для реализации проверки ОЦ требуются значительные затраты времени.

Ограничения целостности в БнД могут задаваться либо при описании структуры таблиц БД (т.е. в схеме БД), либо в программах обработки данных. Первый подход предпочтительнее и не только потому, что описательный (декларативный) способ задания ОЦ представляет собой более высокий уровень контроля, но и потому, что заданные ограничения будут контролироваться при выполнении всех операций над данными.

Разные СУБД обладают различным набором средств для обеспечения целостности данных. Так, некоторые реляционные СУБД поддерживают концепцию первичного ключа, домена и внешнего ключа. При этом соответствующие проверки ОЦ выполняются автоматически. В некоторых системах при описании структуры БД для поля можно задать запрет содержать пустое значение (понятие NOT NULL), можно определить диапазон допустимых значений и другие ОЦ.

При проектировании БнД необходимо изучить, какие возможности по контролю целостности предоставляет используемая СУБД. Если СУБД автоматически не поддерживает нужное ограничение, то обеспечение его соблюдения становится заботой проектировщика.

Нарушение целостности данных может возникнуть в результате ошибок при вводе данных или выполнении иных корректировок, а также в результате сбоев в работе вычислительной системы.

Наличие в БД главных и подчиненных таблиц приводит к дублированию данных, а любое явное или неявное дублирование требует специальных мер по обеспечению целостности данных.

Реализация контроля за соблюдением ОЦ требует в некоторых случаях больших затрат. Если это возможно, то надо задавать условия, когда может быть нарушена целостность, и проверять эти условия только в случаях, которые могут привести к нарушению ОЦ. Например, если имеется условие, что зарплата руководителя не может быть ниже зарплаты подчиненного ему служащего, то увеличение зарплаты руководителя не может нарушить целостность и проверку БД в этом случае проводить не надо.

6.2. Описание объектов и их свойств

Инфологическая модель предметной области может описываться как аналитическими, так и графическими средствами. Графическое представление является наиболее наглядным и простым для восприятия и анализа. Кроме того, графическое представление поддерживается системами автоматизации проектирования БД.

Поэтому для описания объектов и их связей мы воспользуемся графическим способом отображения, позволяющим построить модель, условно называемую «объект – свойство – связь».

При исследовании и анализе ПО заказчик или разработчик выделяет классы объектов. Классом объектов называют совокупность объектов, обладающих одинаковым набором свойств. Например, если в качестве предметной области рассмотреть работу приемной комиссии в институте, то в ней можно выделить следующие классы объектов: абитуриенты, экзаменаторы, аудитории и т.д. Объекты могут быть материальными, как перечисленные выше, или абстрактными, как, например, предметы, по которым абитуриенты сдают экзамены.

Каждый объект представляется своим уникальным идентификатором, по значению которого один объект класса отличается от другого объекта этого же класса. Каждый класс объектов представляется именем этого класса. Так, для объектов класса, именуемого «ПРЕДМЕТ», уникальным идентификатором каждого объекта будет название предмета (например, физика).

Каждый объект обладает определенным набором свойств. Для объектов одного класса набор этих свойств одинаков, а значения свойств могут различаться. Например, все объекты класса «АБИТУРИЕНТ» имеют такие свойства, как «Регистрационный номер», «ФИО», «Год рождения», «Пол» и др.

При описании ПО надо изобразить каждый существенный класс объектов и набор свойств, фиксируемый для объекта данного класса. Абстрактный объект, являющийся обобщенным представителем класса изображается прямоугольником, в котором записывается имя класса (например, АБИТУРИЕНТ). Свойство объекта изображается пунктирным прямоугольником, в котором записано название свойства (например, регистрационный номер).

Каждому классу объектов в ИЛМ присваивается уникальное имя. Именем класса объектов является грамматический

оборот существительного (т.е. существительное, у которого могут быть прилагательные и предлоги). Если имя состоит из нескольких слов, то первым должно стоять существительное, употребляемое в единственном числе. Например, правильным именем класса объектов будет «ТОВАР ПРОДОВОЛЬСТВЕННЫЙ», а не «ПРОДОВОЛЬСТВЕННЫЕ ТОВАРЫ».

Если в ПО традиционно используются разные имена для обозначения какого-либо класса объектов или различные названия свойств объектов (т.е. имеет место синонимия), то все они должны быть зафиксированы в ИЛМ в виде лингвистических отношений. Затем одно из имен или названий выбирается за основное, и только это должно в дальнейшем использоваться в ИЛМ.

При построении ИЛМ желательно дать словесную интерпретацию каждому понятию, особенно если возможно его неоднозначное толкование.

При описании ПО надо отразить связи между объектом и характеризующими его свойствами. Эти связи изображаются в виде линий, соединяющих обозначения объекта и его свойств (рис. 6.3).

Изображение связи между объектом и его свойством учитывет специфику этого свойства.

Так, объект может обладать только одним значением какого-то свойства. Например, каждый человек имеет только одну дату рождения. Такие свойства называются единичными.

Для других объектов возможно одновременное существование нескольких значений у одного и того же свойства объекта. Например, студент может изучать несколько иностранных языков, поэтому свойство «Иностранный язык» будет множественным.

Связь объекта и единичного свойства обозначается одинарной стрелкой, а связь объекта и множественного свойства — двойной стрелкой.

Кроме того, некоторые свойства являются постоянными, их значение не может измениться с течением времени (например, год рождения). Такие свойства называются статическими. Свойства, значения которых могут изменяться со временем, называются динамическими.

Другой особенностью свойства является присутствие этого свойства у всех объектов данного класса либо отсутствие у некоторых объектов. Например, отдельным студентам могут предоставляться один или несколько грантов разными организациями, а другие студенты могут грантов не иметь. Такое свойство называется условным.

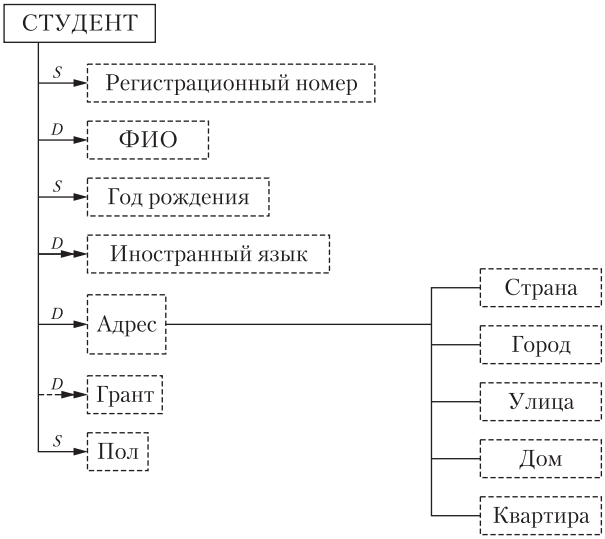


Рис. 6.3. **Изображение связи «объект — свойство»**

Связь условного свойства с объектом изображается пунктирной линией, а для обозначения динамических и статических свойств используются буквы *D* и *S* над соответствующей линией.

Составное свойство (например, адрес) в ИЛМ обозначается квадратом, из которого выходят линии, соединяющие его с элементами составного свойства.

Если в ИЛМ необходимо учесть не только свойства, присущие отдельным объектам класса, но и какие-то интегральные свойства, относящиеся ко всему классу в целом, то в ИЛМ для изображения класса объектов можно использовать какое-нибудь специфическое обозначение, например показанное на рис. 6.4.

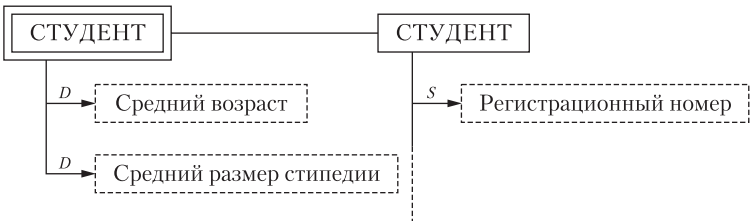


Рис. 6.4. **Изображение класса объектов и интегральных свойств**

6.3. Описание связей между объектами

Помимо связи между объектом и его свойствами в ИЛМ фиксируются связи между объектами разных классов. Различают типы связей «один к одному» (1 : 1), «один ко многим» (1 : *N*), «многие к одному» (*N* : 1) и «многие ко многим» (*N* : *M*). тип связи называют также степенью связи.

Кроме степени связи в ИЛМ для характеристики связи между объектами разных классов надо указывать так называемый класс принадлежности. Класс принадлежности показывает, должен или не должен объект одного класса участвовать в связи с каким-либо объектом другого класса. Класс принадлежности объекта должен быть либо обязательным, либо необязательным.

Если все объекты данного класса должны участвовать в связи, то класс принадлежности для этих объектов называется обязательным. Если некоторые объекты данного класса могут не участвовать в связи, то класс принадлежности называется необязательным.

В ИЛМ связи между объектами с учетом степени связи и класса принадлежности описываются с помощью ER-диаграмм, впервые предложенных Ченом (Chen P. P. S.) в 1979 г., называемых также диаграммами «сущность — связь» или «объект — связь» (ER — первые буквы английских слов Entity (сущность, объект) и Relationship (связь)).

Предположим, что предметной областью является кафедра института, преподаватели которой читают дисциплины, имеющиеся в учебном плане.

Двумя главными классами объектов, или сущностями, представляющими интерес, являются ПРЕПОДАВАТЕЛЬ и КУРС. Эти две сущности соотносятся с помощью связи ЧИТАЕТ, что позволяет сказать:

ПРЕПОДАВАТЕЛЬ ЧИТАЕТ КУРС

Связь ЧИТАЕТ, существующая между указанными сущностями, может быть представлена двумя способами. На рис. 6.5 связь сущностей представлена с помощью диаграммы ER-экземпляров. На диаграмме показано, какой конкретно курс читает каждый преподаватель, причем каждый преподаватель идентифицируется номером преподавателя (НП), а каждый курс — номером курса (НК).

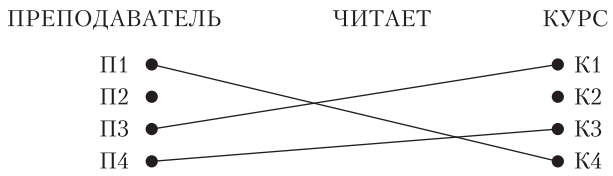


Рис. 6.5. **Пример диаграммы ER-экземпляров**

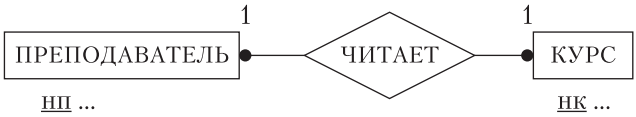


Рис. 6.6. **Пример диаграммы ER-типа**

На рис. 6.6 связь сущностей представлена диаграммой ER-типа.

Приведенные примеры диаграмм отражают степень связи и класс принадлежности объектов, или сущностей, а именно: степень связи объектов, или сущностей, 1 : 1, а классы принадлежности обеих сущностей необязательные.

При изображении диаграмм используется термины «сущность», «связь» и «атрибут», которые были определены в разд. 1.5.

На диаграмме ER-экземпляров (см. рис. 6.5) названия всех сущностей помещены над экземплярами этих сущностей, и в названиях использованы прописные буквы, в то время как каждый экземпляр сущности идентифицируется значением атрибута. Так, КУРС является сущностью, а К1 — конкретным экземпляром сущности.

Связь также именуется, и ее название, составленное из прописных букв, размещается над экземплярами связи, при этом экземпляр каждой отдельной связи задается линией между теми двумя экземплярами сущностей, которые эта связь соединяет. Экземпляр связи между К1 и П3, например, означает, что преподаватель с номером П3 читает курс с номером К1.

Атрибут или набор атрибутов, используемый для идентификации экземпляра сущности, называется ключом сущности.

Каждый экземпляр связи однозначно определяется набором ключей сущностей, соединяемых этой связью.

На диаграмме ER-типа (см. рис. 6.6) сущности представляются в виде прямоугольников, а связи — в виде ромбов. Ниже каждой сущности размещается подчеркнутый атрибут или набор атрибутов, являющийся ключом сущности для этой сущности. Цифры «1» рядом с сущностями указывают степень связи 1 : 1, а точки на линиях, соединяющих ромб с прямоугольниками, обозначают необязательный класс принадлежности.

Можно рекомендовать более простой способ изображения связи на диаграмме ER-типа, учитывающий степень связи сущностей: вместо ромба используется двусторонняя стрелка, соединяющая сущности, а название связи записывается над стрелкой или не указывается. На рис. 6.7 приведены возможные способы изображения связей, соответствующие различным степеням связи.

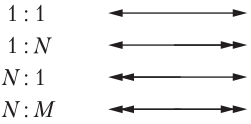


Рис. 6.7. **Альтернативные способы изображения связей сущностей с учетом степеней связи**

Диаграммы, изображенные на рис. 6.5 и 6.6, соответствуют случаю, когда каждый преподаватель читает не более одного курса, и каждый курс читается не более чем одним преподавателем, т.е. допускается наличие преподавателей, не читающих курсы, и курсов, не читаемых вовсе. Таким образом, ни один преподаватель не должен читать более одного курса, и ни один курс не должен читаться более чем одним преподавателем.

Диаграмма ER-типа, изображенная на рис. 6.8, *а*, соответствует случаю, когда каждый преподаватель читает только один курс, а каждый курс читается не более чем одним преподавателем. Поскольку все преподаватели участвуют в чтении какого-либо курса, т.е. связаны с каким-либо курсом, то класс принадлежит сущности ПРЕПОДАВАТЕЛЬ является обязательным и этот факт отмечается точкой в блоке, смежном с прямоугольником сущности.

Диаграмма ER-типа, изображенная на рис. 6.8, *б*, соответствует случаю, когда каждый преподаватель читает не более одного курса (т.е. 1 или 0 курсов), а каждый курс читается только одним преподавателем.

Диаграмма ER-типа, изображенная на рис. 6.8, *в*, соответствует случаю, когда каждый преподаватель читает только один курс и каждый курс читается только одним преподавателем.

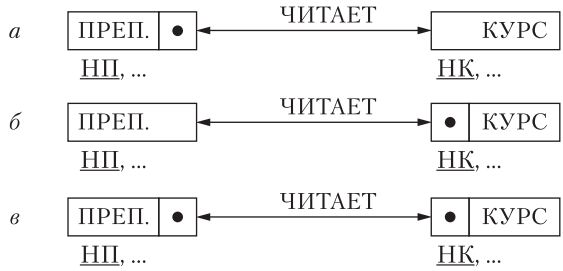


Рис. 6.8 **Диаграммы ER-типа для случая степени связи 1 : 1 и различных классов принадлежности**

Во всех четырех случаях степень связи равна 1 : 1, а сочетание классов принадлежности сущностей различается.

На диаграммах ER-типа непосредственно под каждой сущностью выписывается и выделяется подчеркиванием ключ этой сущности: НП (номер преподавателя) для сущности ПРЕПОДАВАТЕЛЬ и НК (номер курса) для сущности КУРС. Точки, расположенные вслед за каждым из этих атрибутов, указывают на то, что никакие другие атрибуты, сущности не могут быть частью ее ключа.

Помимо рассмотренных случаев организации чтения курсов преподавателями возможны и другие, например:

1) каждый преподаватель может читать одновременно несколько курсов, но каждый курс читается не более чем одним преподавателем;

2) каждый преподаватель читает не более одного курса, но каждый курс может читаться сразу несколькими преподавателями;

3) каждый преподаватель может читать несколько курсов (в частности, ни одного курса) и каждый курс может читаться несколькими преподавателями (в частности, ни одним преподавателем).

Этим случаям соответствуют степени связи $1 : N$, $N : 1$, $N : M$ и различные сочетания классов принадлежности сущностей. На рис. 6.9—6.11 приведены диаграммы ER-типа, отражающие все возможные случаи организации чтения курсов преподавателями.

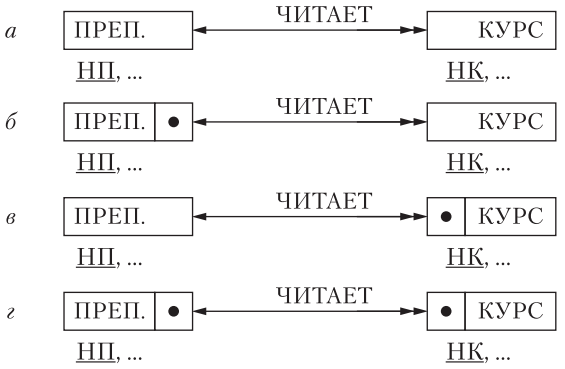


Рис. 6.9. **Диаграммы ER-типа для случая степени связи 1 : *N***

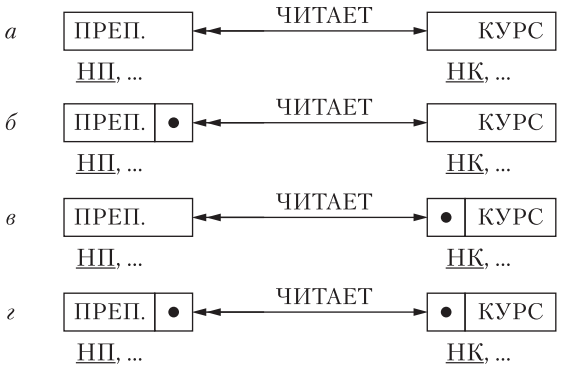


Рис. 6.10. **Диаграмма ER-типа для случая степени связи *N* : 1**

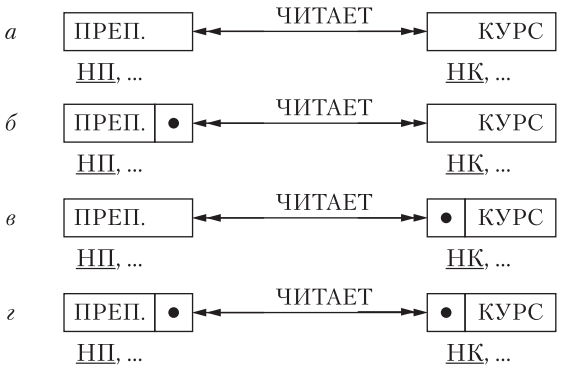


Рис. 6.11. **Диаграмма ER-типа для случая степени связи *N* : *M***

6.4. Описание сложных объектов

Ранее объекты рассматривались без учета их сложности. По степени сложности объекты могут быть простые и сложные. Объект считается простым, если он рассматривается как неделимый. Сложный объект представляет собой объединение других объектов, также выделяемых в ПО.

Различают несколько разновидностей сложных объектов: составные, обобщенные и агрегированные.

Составной объект соответствует отображению отношения «целое — часть», например: УЗЛЫ — ДЕТАЛИ, КЛАСС — УЧЕНИКИ и т.п.

Для представления составных объектов в ИЛМ используются диаграммы ER-типа:

Обобщенный объект отражает наличие связи «род — вид» между объектами ПО. Например, объекты СТУДЕНТ, АСПИРАНТ образует обобщенный объект УЧАЩИЙСЯ.

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

Определение родовидовых связей означает классификацию объектов ПО по тем или иным признакам.

Подклассы могут выделяться в ИЛМ в явном виде, для чего при графическом изображении используются специальные обозначения подкласса, например, треугольником. На рис. 6.12 показан фрагмент ИЛМ, представляющий обобщенный объект ЛИЧНОСТЬ для института. Для этого выделено несколько категорий: СОТРУДНИК, СТУДЕНТ, АСПИРАНТ.

Агрегированный объект обычно соответствует какому-либо процессу, в который оказываются вовлеченными другие объекты. Агрегированный объект именуется отглагольным

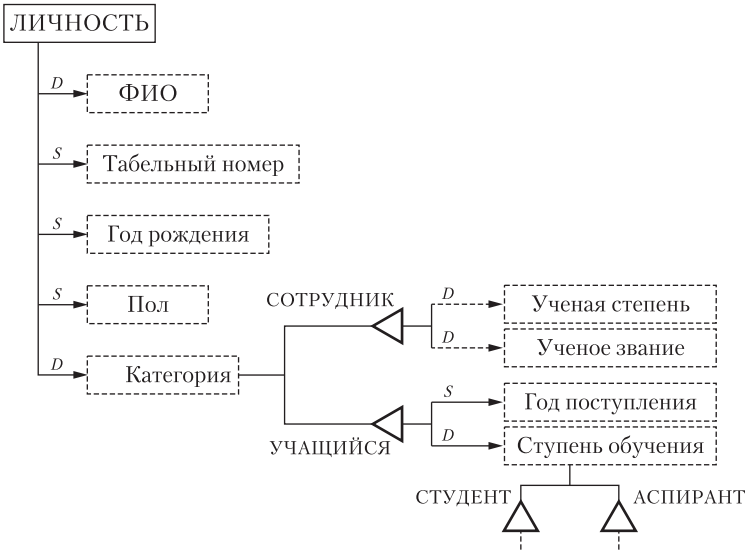


Рис. 6.12. **Изображение обобщенного объекта**

существительным (например, поставлять — поставка, выпускать — выпуск, продавать — продажа и т.д.).

В ИЛМ агрегированный объект изображается ромбом, в котором указано имя объекта. Этот ромб соединяется линиями с условными обозначениями объектов, которые образуют агрегированный объект. Свойства агрегированного объекта изображаются пунктирным прямоугольником. Например, агрегированный объект ПОСТАВКА (рис. 6.13) объединяет

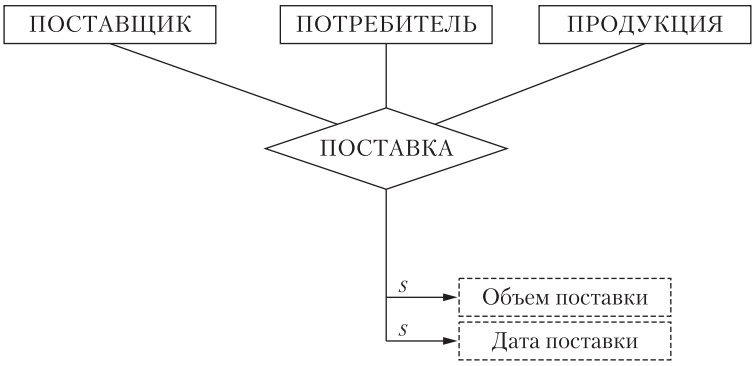


Рис. 6.13. **Изображение агрегированного объекта**

в себе объекты ПОСТАВЩИК, который поставляет продукцию, ПОТРЕБИТЕЛЬ, который получает эту продукцию, саму поставляемую ПРОДУКЦИЮ.

6.5. Даталогическое проектирование

6.5.1. Общие сведения

Цель даталогического проектирования заключается в создании ДЛМ, которая отображает логические связи между элементами данных безотносительно к их смысловому содержанию и среде хранения. Эта модель строится в терминах информационных единиц, предусмотренных в конкретной СУБД. На рис. 6.14 показана последовательность разработки ДЛМ.

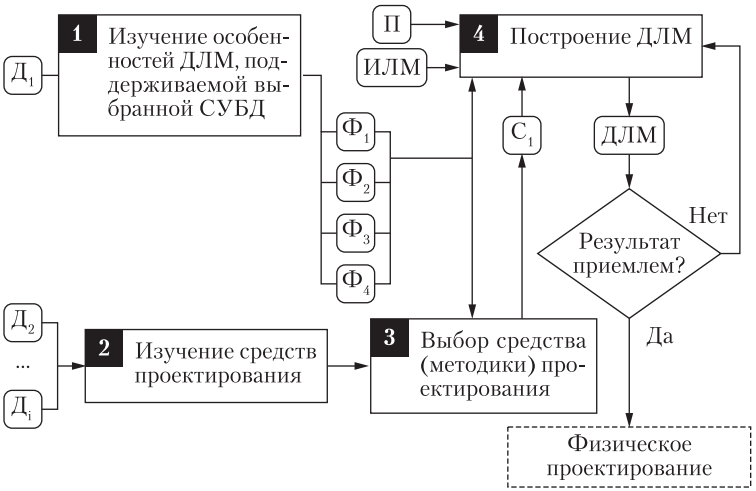


Рис. 6.14. **Последовательность разработки ДЛМ**
($Д_1$ — документация по СУБД; $Д_2$, ..., $Д_i$ — документация по средствам проектирования; П — перечень хранимых показателей (атрибутов); ДЛМ — даталогическая модель; ИЛМ — инфологическая модель; $С_1$ — средство проектирования; факторы, влияющие на ДЛМ и выбор средства проектирования: $Ф_1$ — набор допустимых даталогических конструкций; $Ф_2$ — операторы ЯОД; $Ф_3$ — ограничения, налагаемые СУБД на ДЛМ; $Ф_4$ — возможности физической организации данных)

Любая СУБД оперирует с допустимыми для нее логическими единицами данных, а также допускает использование определенных правил композиции логических структур более высокого уровня и составляющих информационных единиц более низкого уровня. Кроме того, многие СУБД накладывают количественные и иные ограничения на структуру БД. Поэтому, прежде чем приступить к построению ДЛМ, необходимо детально изучить особенности СУБД, определить факторы, влияющие на выбор проектного решения, ознакомиться с существующими методиками проектирования, а также провести анализ имеющихся средств автоматизации проектирования, оценить возможность и целесообразность их использования.

Хотя даталогическое проектирование является проектированием логической структуры БД, на него оказывают влияние возможности физической организации данных, предоставляемые конкретной СУБД. Поэтому знание особенностей физической организации данных может быть полезным при проектировании логической структуры.

Логическая структура БД, а также сама заполненная данными БД являются отображением реальной ПО. Поэтому на выбор проектных решений самое непосредственное влияние оказывает специфика ПО, отраженная в ИЛМ.

Результатом даталогического проектирования является описание логической структуры БД на ЯОД. В спроектированной логической структуре БД должны быть определены все информационные единицы и связи между ними, заданы имена информационных единиц, их тип и количественные характеристики (например, длина поля).

6.5.2. Подход к даталогическому проектированию

Процесс проектирования БД предусматривает предварительную классификацию объектов ПО, систематизированное представление информации об объектах и связях между ними в ИЛМ.

На начальных этапах даталогического проектирования должен быть определен состав БД.

При проектировании логической структуры БД осуществляются преобразование исходной ИЛМ в модель данных, поддерживаемую конкретной СУБД, и проверка адекватности полученной ДЛМ отображаемой ПО.

Для любой ПО существует множество вариантов проектных решений ее отображения в ДЛМ. Методика проектирования должна обеспечивать выбор наиболее подходящего проектного решения.

Минимальная логическая единица данных для всех СУБД семантически одинакова и соответствует либо идентификатору объекта, либо свойству объекта или процесса.

Связи между объектами ПО, отраженные в ИЛМ, могут отображаться в ДЛМ либо посредством совместного расположения соответствующих им информационных элементов, либо путем объявления связи между ними. Не все виды связей, существующие в ПО, могут быть непосредственно отображены в конкретной ДЛМ. Так, многие СУБД не поддерживают непосредственно степень связь N: M между объектами. В этом случае в ДЛМ вводится дополнительный вспомогательный элемент, отображающий эту связь.

Следует иметь в виду, что связи, имеющиеся в ПО и отраженные в ИЛМ, могут быть представлены не только посредством структуры БД, но и программным путем. Например, при отображении обобщенных объектов можно не выделять подклассы на уровне логической структуры БД, а предусмотреть выделение подклассов программным путем при обработке хранимых данных.

Принимаемое проектное решение зависит не только от специфики отображаемой ПО, но и от характера обработки информации, хранимой в БД. Например, рекомендуется хранить вместе информацию, часто обрабатываемую совместно, и, наоборот, разделять по разным файлам информацию, не использующуюся одновременно. Информацию, используемую часто, и информацию, частота обращения к которой мала, также следует хранить в разных файлах.

6.5.3. Определение состава БД

При переходе от ИЛМ к ДЛМ следует иметь в виду, что ИЛМ включает в себя всю информацию, необходимую и достаточную для проектирования БД. Но это не означает, что все свойства, зафиксированные в ИЛМ, должны в явном виде отражаться в ДЛМ. Прежде чем строить ДЛМ, необходимо решить, какая информация будет храниться в БД. Например, ИЛМ может содержать вычисляемые показатели, но вовсе не обязательно хранить их в БД.

Один из подходов к определению состава показателей, хранимых в БД, основан на принципе синтезирования: в БД должны храниться только исходные показатели, все производные показатели должны вычисляться в момент выполнения запроса.

Достоинства такого подхода:

1) простота и однозначность в принятии решения о том, «что хранить»;

2) отсутствие явного дублирования информации со всеми из этого последствиями (меньше объем памяти, проще проблемы контроля целостности БД);

3) потенциальная возможность получить любой расчетный показатель, а не только те, которые хранятся в БД.

Несмотря на эти достоинства, в каждом конкретном случае нужно оценивать целесообразность хранения вычисляемых показателей в БД с учетом частоты использования и трудоемкости вычисления этих показателей.

При отображении объекта в БД идентификатор объекта будет атрибутом, который в большинстве случаев используется для однозначной идентификации объекта. Однако может появиться необходимость введения искусственных идентификаторов или кодов. Это нужно, когда:

1) в ПО наблюдается омонимия, т.е. естественный идентификатор объекта не обладает уникальностью. Например, среди студентов могут быть однофамильцы и полные тезки. В этом случае для обеспечения однозначной идентификации объектов необходимо использовать коды;

2) если объект участвует во многих связях, то для идентификации связи удобнее использовать короткий код объекта, чем длинный естественный идентификатор объекта;

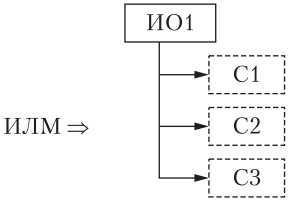
3) если естественный идентификатор может со временем изменяться, то при отсутствии кода это может вызвать много проблем с поиском нужных сведений.

6.6. Метод проектирования реляционной базы данных на основе ИЛМ

Рассмотрим метод проектирования реляционной БД, основанный на анализе ИЛМ и переходе от нее к отношениям ДЛМ. Этот метод является достаточно простым и наглядным и в то же время дает хорошие результаты.

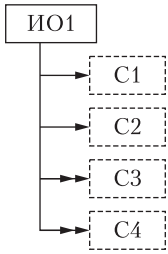
Для перехода от ИЛМ к реляционной ДЛМ используются следующие правила, в которых идентификаторы объектов обозначены как ИО1, ИО2, ИО3, а прочие свойства объектов — как С1, С2 и т.д.

1. Для каждого простого объекта и его единичных свойств строится отношение, атрибутами которого являются идентификатор объекта и реквизиты, соответствующие каждому из единичных свойств:



R1 (ИО1, С1, С2, С3) ⇐ ДЛМ

2. Если у объекта имеются множественные свойства, то каждому из них ставится в соответствие отдельное отношение. Ключом этого отношения будет идентификатор соответствующего объекта и реквизит, отражающий множественное свойство:

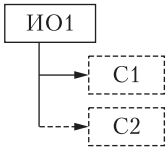


R1 (ИО1, С1, С2)
R2 (ИО1, С3)
R3 (ИО1, С4)

3. Если между объектом и его свойством имеется условная связь (условное свойство), то при отображении в ДЛМ возможны следующие варианты:

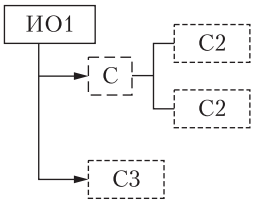
а) если многие из объектов обладают условным свойством, то его можно хранить в БД так же, как и обычное свойство;

б) если только незначительное число объектов обладает условным свойством, то можно выделить отношение, которое будет включать идентификатор объекта и атрибут, соответствующий условному свойству. Это отношение будет содержать столько строк, сколько объектов имеют условное свойство.



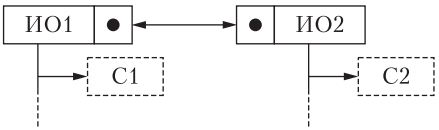
R1 (ИО1, С1, С2)
или
R1 (ИО1, С1)
R1 (ИО1, С2)

4. Если объект имеет составное свойство, то оно представляется в отношении набором атрибутов, соответствующих элементам составного свойства:



R1 (ИО1, С1, С2, С3)

5. Если между объектами имеется степень связи 1 : 1, то ДЛМ определяется классом принадлежности объектов. Если класс принадлежности обоих объектов является обязательным, то ДЛМ задается одним отношением, в котором атрибутами будут идентификаторы объектов и свойства обоих объектов:



R1 (ИО1, С1, ..., ИО2, С2, ...)

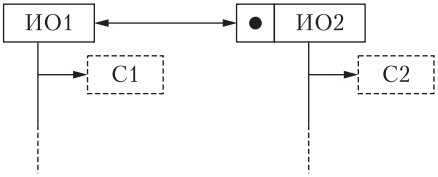
Такая модель потребует меньше всего памяти. Однако если в запросах часто требуется информация отдельно по каждому из объектов, то для ускорения поиска информации целесообразно каждый объект представить отдельным отношением, а связь объектов представить в ДЛМ, указав идентификатор одного объекта в качестве атрибута в отношении, соответствующем другому объекту, например:

R1(ИО1, С1, ИО2)

R1(ИО2, С2)

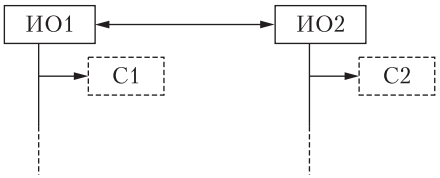
Если класс принадлежности одного из объектов необязательный, то ДЛМ задается двумя отношениями, причем идентификатор объекта, для которого класс принадлежности

является необязательным, добавляется в качестве атрибута в отношение, соответствующее объекту с обязательным классом принадлежности:



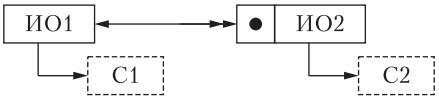
R1 (ИО1, С1, ...)
R2 (ИО2, С2, ..., ИО1)

Если класс принадлежности обоих объектов является необязательным, то ДЛМ задается тремя отношениями: по одному для каждого объекта и одно для отображения связи между объектами:



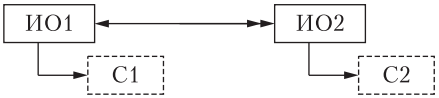
R1 (ИО1, С1, ...)
R2 (ИО2, С2, ...)
R3 (ИО1, ИО2)

6. Если между объектами имеется степень связи 1 : *M* и класс принадлежности многосвязного объекта является обязательным, то независимо от класса принадлежности односвязного объекта ДЛМ задается двумя отношениями, по одному для каждого объекта. Отношение, соответствующее многосвязному объекту, нужно дополнить атрибутом, являющимся идентификатором (ключом) односвязного объекта:



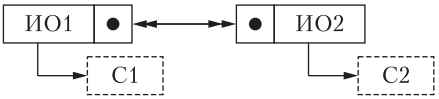
R1 (ИО1, С1)
R2 (ИО2, С2, ИО1)

Если класс принадлежности многосвязного объекта является необязательным, то ДЛМ задается тремя отношениями: по одному для каждого объекта и одно для отображения связи между объектами. Связь объектов задается атрибутами, являющимися идентификаторами (ключами) объектов:



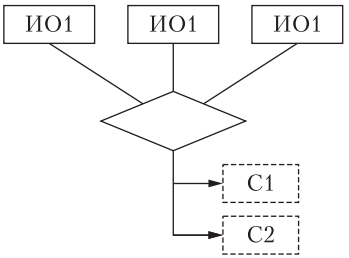
R1 (ИО1, С1)
R2 (ИО2, С2
R3 (ИО1, ИО2)

7. Если между объектами ПО имеется степень связи $M:N$, то ДЛМ задается тремя отношениями независимо от класса принадлежности объектов:



R1 (ИО1, С1)
R2 (ИО2, С2)
R3 (ИО1, ИО2)

8. Каждому агрегированному объекту, имеющемуся в ПО, в ДЛМ соответствует отдельное отношение. Атрибутами этого отношения будут идентификаторы всех объектов, входящих в агрегированный объект, а также свойства этого объекта:

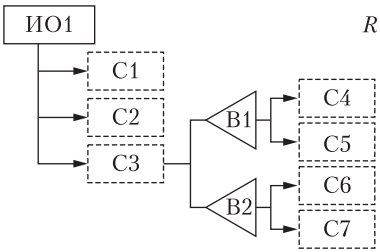


R1 (ИО1, ИО2, ИО3, С1, С2)

9. При отображении обобщенных объектов возможны разные решения. Во-первых, всему обобщенному объекту может быть поставлено в соответствие одно отношение. Во-вторых, каждой категории объектов нижнего уровня ставится в соответствие отдельное отношение.

В первом случае атрибутами отношения будут все единичные свойства, присущие объектам хотя бы одной категории, плюс идентификатор объекта.

Во втором случае каждое отношение включает в себя идентификатор объекта, те свойства, которые присущи объектам данной категории, а также свойства, которыми обладают родовые объекты, стоящие выше его по иерархии.



R1 (ИО1, С1, С2, С3, С4, С5, С6, С7)

или

R1 (ИО1, С1, С2, С4, С5)
R2 (ИО1, С1, С2, С6, С7)

Кроме этих двух крайних решений, возможны и комбинированные варианты. Выбор конкретного решения будет зависеть от многих факторов, в том числе насколько часто информация о разных категориях объектов обрабатывается совместно, как велико различие в «видовых» свойствах.

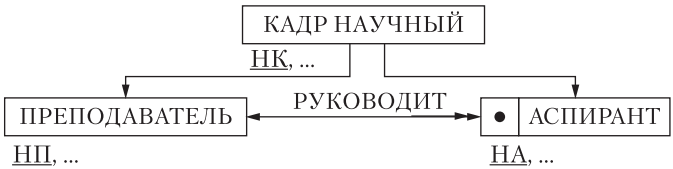
10. Составной объект, для которого характерно наличие связи «целое — часть», также может быть отображен в ДЛМ по-разному. Например, между объектами УЗЕЛ и ДЕТАЛЬ имеется связь $M : N$, так как одна и та же деталь может входить в разные узлы и, наоборот, в узел входят разные детали. Состав узла обычно является сложным, и отражать его в явном виде в структуре БД нежелательно, а иногда и невозможно. Кроме того, рассматриваемая связь реализована на однородном множестве объектов. В этом случае для отображения связи «целое — часть» можно использовать два отношения. Первое из них будет содержать информацию о самих объектах, а второе — информацию о связи между ними, а также дополнительную информацию, характеризующую эту связь. Для состава узла это могут быть атрибуты «Что входит», «Куда входит», «Количество».

Связь «целое — часть» может отражать, например, структуру какой-то организации. В этом случае, скорее всего, степень связи будет $1 : M$, и для построения ДЛМ можно использовать рекомендации правила 6.

11. В некоторых случаях одних объектов (сущностей) и связей может оказаться недостаточно для всестороннего моделирования предметной области. Один из таких случаев возникает тогда, когда экземпляры некоторой сущности должны играть разные роли в деятельности организации.

В качестве примера предположим, что для кафедры института необходимо хранить информацию о процессе подготовки научных кадров. Различают две категории объектов в этом процессе: преподаватели и аспиранты. И те и другие являются научными кадрами, но играют разные роли в процессе подготовки научных кадров: преподаватели передают свои знания, а аспиранты приобретают знания.

С учетом ролей диаграмма ER-типа будет иметь следующий вид:



При разработке ДЛМ можно следовать такому правилу. Исходная сущность представляется одним отношением, причем ключ сущности служит первичным ключом. Ролевые объекты и связи, их соединяющие, представляются в ДЛМ таким числом отношений, которое определяется ранее описанными правилами, причем каждая роль трактуется как обычная сущность.

Согласно этому правилу для нашего примера ДЛМ задается тремя отношениями:

R1(НК, ...)

R2(НП, ...)

R3(НА, ..., НП)

В этих отношениях множества значений атрибутов НП и НА не пересекаются (НП $\cap$ НА = $\varnothing$), а множество значений атрибута НК состоит из значений атрибутов НП и НА (НК = = НП $\cup$ НА).

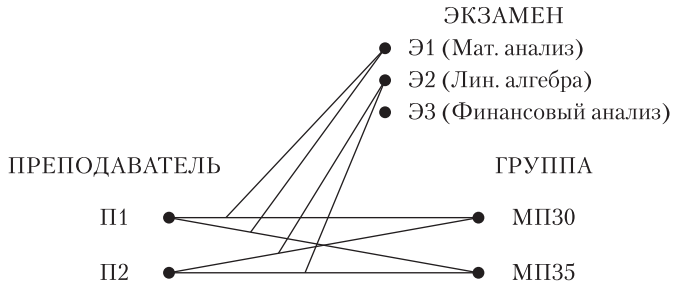
В отношение *R1* помимо атрибута НК попадают атрибуты, общие для преподавателей и аспирантов, а в отношения *R2* и *R3* помимо атрибутов НП и НА соответственно попадают атрибуты, специфичные для преподавателей и аспирантов.

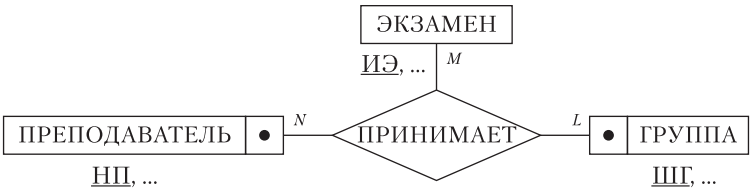
Для увеличения скорости доступа при выполнении запросов в отношении *R1* можно предусмотреть дополнительный атрибут для указания роли объекта. Этот атрибут позволит избежать поиска в отношениях *R2* и *R3*, необходимого для определения рода деятельности объекта.

Полученная таким способом ДЛМ напоминает результат применения правила 9 для обобщенного объекта. В этом нет ничего странного: разным точкам зрения на одну и ту же ПО соответствуют различные ИЛМ, но отображение этих моделей в ДЛМ может привести к похожим результатам.

12. До сих пор в ИЛМ присутствовали только бинарные связи объектов, т.е. связь между парой объектов. Хотя с помощью бинарных связей могут быть описаны многие ситуации в ПО, тем не менее неизбежно могут возникнуть и такие ситуации, в которых между объектами существуют связи более высокого порядка.

Например, рассмотрим ситуацию приема экзаменов преподавателями в группах, представив ее в виде диаграмм ER-экземпляров и ER-типа:





(НП — номер преподавателя; ИЭ — идентификатор экзамена; ШГ — шифр группы)

В случае трехсторонних связей в ИЛМ даталогическая модель состоит из четырех отношений: по одному для каждой сущности, причем ключ каждой сущности должен служить первичным ключом для соответствующего отношения, и одно отношение для связи сущностей. Отношение, порожденное связью, будет иметь среди своих атрибутов ключи сущностей от каждой сущности.

Для рассматриваемого примера ДЛМ состоит из таких отношений:

R1(НП, ...)

R2(ИЭ, ...)

R3(ШГ, ...)

R4(НП, ИЭ, ШГ, ...)

При наличии *n*-сторонней связи требуется $(n + 1)$ отношение: *n* отношений для сущностей и одно отношение для связи.

Полученная таким образом ДЛМ содержит отношение для связи, которое напоминает результат применения правила 8 для агрегированного объекта. Это не случайно, поскольку ИЛМ отображает один процесс, в который вовлечено не-

сколько объектов, т.е. ПО можно трактовать как агрегированный объект, именуемый ПРИЕМОМ ЭКЗАМЕНОВ.

6.7. Пример проектирования реляционной базы данных на основе ИЛМ

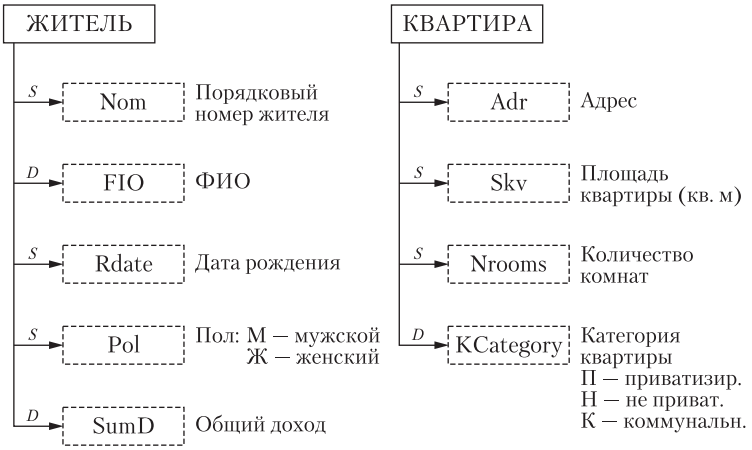
Возьмем в качестве предметной области жителей города Зеленограда и спроектируем простейшую РБД, в которой будут храниться сведения, представляющие интерес для налоговой инспекции.

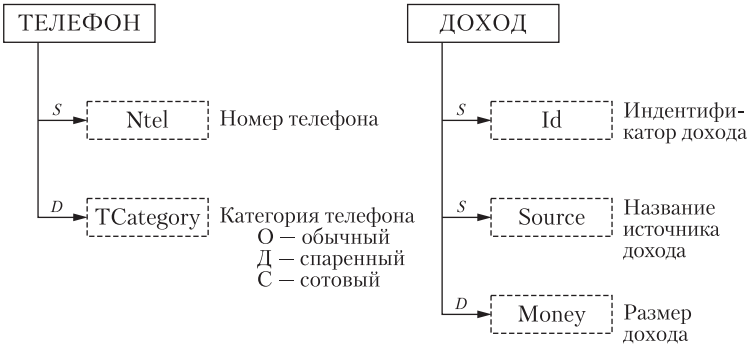
Проектирование начинается с разработки ИЛМ, которая включает в себя следующие компоненты:

1) описание объектов ПО и связей между ними;
2) лингвистические отношения;
3) алгоритмические связи показателей;
4) описание информационных потребностей пользователей;
5) ограничения целостности.

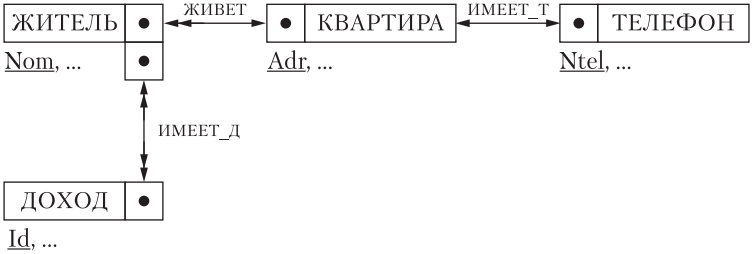
6.7.1. Описание объектов и связей между ними

Объектами, представляющими интерес, будут ЖИТЕЛЬ, КВАРТИРА, ТЕЛЕФОН, ДОХОД. Опишем каждый из них:





Связи между объектами отражаются на диаграмме ER-типа:



6.7.2. Лингвистические отношения

В лингвистических отношениях должно быть дано толкование используемых в ИЛМ терминов и понятий, например:

Nom — уникальный номер жителя Зеленограда, в качестве которого используется целое число;

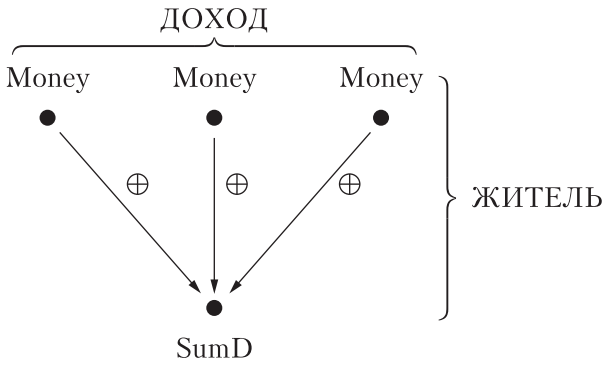
Adr — адрес жителя; задается указанием улицы, номеров дома (корпуса) и квартиры;

Source — обозначение источника дохода: Пенсия1 — пенсия по старости; Пособие1 — пособие на ребенка; Стипендия1 — повышенная стипендия; Работа1 — работа в банке; и т.д.;

Money — размер дохода в рублях.

6.7.3. Алгоритмические связи показателей

Из всех показателей, отраженных в ИЛМ, алгоритмически связанным является общий доход (SumD). Его вычисление описывается следующим графом взаимосвязи показателей:



6.7.4. Описание информационных потребностей пользователей

Здесь должны быть определены все запросы, которые будут поступать от пользователей БД, например:

1) вывести список всех жителей с указанием общего дохода;

2) вывести список жителей, у которых общий доход не меньше налогооблагаемого минимума;

3) подсчитать налоги отдельных жителей и общую сумму налогов.

Описание информационных потребностей служит основой для разработки информационной системы, с помощью которой пользователь будет получать ответы на свои запросы.

6.7.5. Ограничения целостности

Здесь формулируются условия, которым должны удовлетворять отдельные показатели и группы показателей, чтобы информация, хранимая в БД, имела смысл. Например, размер дохода не должен быть отрицательным, источник дохода должен выбираться из известного списка и т.п.

Ограничения целостности используются при разработке прикладной системы, чтобы контролировать правильность данных, вводимых в БД, и корректность вычислений.

После создания ИЛМ переходим к разработке ДЛМ; при этом определим состав БД и воспользуемся ранее сформулированными правилами, учитывающими особенности ИЛМ.

6.7.6. Определение состава БД

Будем хранить в БД все показатели, в том числе и вычисляемый показатель SumD, поскольку его значение требуется при выполнении многих запросов.

6.7.7. Определение отношений, включаемых в БД

Для определения отношений, из которых будет состоять БД, применим соответствующие правила к каждой паре связанных объектов (сущностей), изображенных на диаграмме ER-типа, а затем проанализируем полученные отношения, чтобы сократить их общее количество.

Согласно правилу 6 связанные объекты ЖИТЕЛЬ — КВАРТИРА представляются двумя отношениями:

```
PERSON'(Nom, …, Adr)
FLAT'(Adr, …)
```

Согласно правилу 7 связанные объекты ЖИТЕЛЬ — ДОХОД представляются тремя отношениями:

```
PERSON''(Nom, …)
PROFIT''(Id, …) HAVE_D''(Nom, Id)
```

Согласно правилу 5 связанные объекты КВАРТИРА — ТЕЛЕФОН представляются двумя отношениями:

```
FLAT'''(Adr, …)
TPHONE'''(Ntel, …, Adr)
```

Анализ этих семи отношений позволяет установить, что для отображения ИЛМ ПО в ДЛМ достаточно пяти отношений:

```
PERSON(Nom, FIO, Rdate, Pol, SumD, Adr)
FLAT(Adr, Skv, Nrooms, KCategory)
HAVE_D(Nom, Id)
TPHONE(Ntel, TCategory, Adr)
PROFIT(Id, Source, Money)
```

6.7.8. Описание логической структуры БД на языке СУБД

Даталогическое проектирование завершается описанием логической структуры БД на языке СУБД. Это описание называется схемой БД и помимо всего прочего содержит такие характеристики атрибутов отношений, как тип и длина (размер) атрибута. Для СУБД Paradox схема приведена в табл. 6.1.

Таблица 6.1

Схема базы данных

Таблица БД	Атрибут	Тип	Размер	Допустимые значения	Значение по умолчанию
PERSON	Nom	Autoincrement			
	FIO	Alpha	30		
	Rdate	Date			
	Pol	Alpha	1	М, Ж	
	SumD	Money			0
	Adr	Alpha	30		
FLAT	Adr	Alpha	30		
	Skv	Number		>=0	0
	Nrooms	Short		0..4	0
	KCategory	Alpha	1	П,Н,К	Н
TPHONE	Ntel	Alpha	8	###–####	
	TCategory	Alpha	1	О,Д,С	О
	Adr	Alpha	30		
PROFIT	Id	Autoincrement			
	Source	Alpha	20		
	Money	Money		>=0	0
HAVE_D	Nom	Long Integer		>0	
	Id	Long Integer		>0	

Ввод информации в БД и получение нужной информации из БД осуществляются либо непосредственно средствами СУБД, либо с помощью специально разрабатываемой прикладной системы, использующей команды СУБД.

Посмотрим, как обстоят дела с аномалиями вставки, обновления и удаления в спроектированной РБД по сравнению с однотабличной БД Zgrad (см. табл. 5.1).

Если появляется новый житель (например, новорожденный), у которого отсутствуют источники дохода, то в спроектированной РБД *аномалия вставки* не возникает. Действительно, если у нового жителя отсутствует источник дохода, то информа-

ция о жителе будет занесена только в одно отношение PERSON, причем даже атрибут SumD будет иметь определенное (нулевое) значение, соответствующее действительности.

Аномалия обновления в спроектированной РБД при изменении адреса у конкретного жителя не возникает, поскольку в этом случае будет обновляться атрибут Adr в одном кортеже отношения PERSON и, возможно, появится новый кортеж с описанием адреса и характеристик квартиры в отношении FLAT, если в РБД не было сведений об этой квартире.

Не проявляется аномалия обновления в спроектированной РБД и при изменении номера телефона, установленного в квартире. Если об этом факте сообщит любой из жителей такой квартиры (например, с адресом 801-12), то изменения отразятся только в одном кортеже отношения TPHONE (обновится номер телефона, установленного в квартире 801-12).

Аномалия удаления. В отношении Zgrad присутствует только один кортеж, соответствующий Ильину Ф. П. (см. табл. 5.1). Предположим, налоговый инспектор узнает, что этот житель лишился своего источника дохода, условно именуемого Работа3, и удаляет этот кортеж из отношения. Поскольку это единственный кортеж с информацией об Ильине, его удаление приведет к исключению жителя из БД. Если налоговый инспектор вслед за этим запросит список фамилий, имен и отчеств жителей, зарегистрированных в БД, то Ильина в списке не окажется, хотя он продолжает жить по прежнему адресу.

В спроектированной РБД аномалия удаления не проявляется: для удаления сведений об исчезнувшем у Ильина источнике дохода следует исключить из отношения HAVE_D кортеж, соответствующий этому жителю. После этого сведения о существовании Ильина остаются в отношении PERSON без изменений.

Итак, спроектированная реляционная БД не создает проблем вставки, обновления и удаления при работе с ней.

6.8. Автоматизация проектирования баз данных

6.8.1. CASE-средства и методологии проектирования

Подход, основанный на применении ER-диаграмм, широко используется как при ручном, так и при автоматизированном

проектировании БД, реализуемом с помощью специальных программно-технологических средств, которые принято называть CASE-средствами (CASE — Computer Aided Software Engineering) [2, 3, 6]. К таким средствам относятся Oracle Designer, Sybase PowerDesigner, Silverrun, ER/Studio, Design/IDEF, ERwin и т.д.

CASE-средства позволяют автоматизировать не только проектирование БД, но и другие этапы разработки информационной системы, в частности моделирование процессов и функций, в которых участвуют объекты предметной области, подлежащей информатизации. CASE-средства основаны на методологиях структурного или объектно-ориентированного анализа и проектирования, помогающих формировать спецификации в виде диаграмм или текстов для описания требований заказчика, связей между моделями компонентов проектируемой системы, динамики ее поведения и архитектуры программных средств, которые войдут в ее состав.

Oracle Designer

Oracle Designer (старое название — Designer/2000) предназначен для моделирования сложных систем, анализа и проектирования с использованием диаграмм различного типа [12].

В Designer используется собственная методология проектирования, основанная на структурном подходе к разработке прикладной системы, разбиении жизненного цикла этой системы на этапы с учетом автоматизации перехода от одного этапа к другому. Designer поддерживает единую информационную базу проекта (*репозиторий*), в которой содержатся спецификации проектов на всех этапах разработки и через которую обеспечивается согласованность работы всех разработчиков. Возможность совместной работы с базой проекта решает проблему взаимной координации усилий и существенно способствует успеху крупномасштабных проектов. На всех уровнях проектирования используются приложения с графическим интерфейсом, что увеличивает наглядность и упрощает работу с Designer.

Средства концептуального проектирования включают в себя ER-диаграммы, диаграммы функциональной иерархии, описывающие функции, которые выполняет система, и диаграммы потоков данных, циркулирующих в предметной области.

Структура базы данных, схема взаимодействия программных модулей, структура отдельных процедур или экранных форм прикладной системы отражается в виде графических моделей.

Имеющиеся средства реинжиниринга («перепроектирования») позволяют воссоздать спецификации для анализа поставленной задачи и проектирования с целью модернизации уже существующей системы. В дальнейшем это позволяет дорабатывать и поддерживать модернизированную систему уже с помощью CASE-средств.

Применение CASE-средств существенно увеличивает производительность на всех этапах разработки прикладной системы и значительно сокращает затраты на ее сопровождение. Средства управления проектом, включенные в состав Designer, позволяют контролировать весь процесс разработки, обеспечивая на каждом этапе полноту и непротиворечивость структурных и функциональных спецификаций, по которым создается работающая версия прикладной системы.

Sybase PowerDesigner

Sybase PowerDesigner является средством моделирования и проектирования современных прикладных систем, позволяющим осуществлять анализ и моделирование процессов и структур баз данных [6, 7, 10].

Основные возможности PowerDesigner включают в себя:

1) моделирование процессов: PowerDesigner позволяет пользователям, не обладающим техническим опытом, но знающим предметную область, описывать процессы в рамках удобной графической модели; также поддерживается генерация программного кода для разрабатываемой прикладной системы;

2) моделирование данных: проектирование и генерация структур базы данных на основе концептуального и даталогического проектирования; поддерживаются более 45 широко распространенных на рынке СУБД, включая Oracle, Sybase Adaptive Server, DB2, SQL Server и др., с возможностью генерации структуры БД как по модели, так и восстановления модели по существующей структуре (reverse engineering); предусмотрены средства проектирования хранилищ данных [13];

3) реализован импорт моделей, созданных с помощью программы ERwin: инфологические и даталогические модели из ERwin могут быть импортированы и преобразованы в модели PowerDesigner с полным сохранением метаданных и связей между моделями;

4) объектное моделирование приложений: полная поддержка анализа и проектирования в рамках методологии UML [17], основанная на работе с различными типами UML-диаграмм (прецедентов (use case), активности (activity), последовательностей (sequence), классов (class), компонентов (component), взаимодействия (collaboration), состояния (statechart), объектов (object), развертывания (deployment)); на базе диаграммы классов реализована возможность генерации программного кода на языках Java, XML, C++, C#, PowerBuilder, Visual Basic.

Silverrun

Silverrun обеспечивает поддержку различных методологий, основанных на раздельном построении функциональной и информационной моделей (диаграмм потоков данных DFD (Data Flow Diagram) и ER-диаграмм) [2]. Настройка на конкретную методологию обеспечивается выбором требуемой графической нотации моделей и набора правил проверки проектных спецификаций. В системе имеются готовые настройки для наиболее распространенных методологий: DATARUN (основная методология, поддерживаемая Silverrun), Гейна — Сарсона, Йодана — Де Марко, Уорда — Меллора, Information Engineering (IE, или «лучеобразное» представление) [9].

Методология DATARUN регламентирует следующую последовательность этапов разработки:

1) построение модели предметной области: создаются функциональная (DFD) и информационная (ER) модели предметной области, причем при построении функциональной модели выявляются первичные структуры данных, которые преобразуются в сущности ER-диаграммы; в результате получается модель процессов, содержащая первичные структуры данных, и инфологическая модель;

2) построение архитектуры информационной системы: на основе анализа существующих систем принимается решение, из каких приложений (подсистем) будет состоять проектируемая система; архитектура системы документируется в виде DFD, где функции представляют компоненты приложений

с указанием используемой информации посредством ссылки на сущности и связи в ER-диаграмме, которая делится на группы сущностей, обрабатываемых приложениями;

3) проектирование приложений (подсистем): на основе инфологической модели строится даталогическая модель; части этой модели, соответствующие различным приложениям, оформляются как подсхемы базы данных; для каждого приложения создается детальный проект, в котором определяются модель системных процессов (программных функций) и подсхемы базы данных для каждой функции (спецификация интерфейса); поскольку все приложения работают с подсхемами одной базы данных, то достигается их совместное функционирование;

4) создание приложений: на основе даталогической модели генерируется код (например, на языке SQL) для ее создания на сервере баз данных; программируются системные процессы с учетом возможного их разнесения по узлам распределенной системы (часть процессов реализуется как хранимые процедуры на сервере, часть — как сервисы монитора транзакций, часть — как программы клиентской части); интерфейс приложений (обычно составляющий до 70—80% всей системы) может быть быстро создан перенесением соответствующей ему подсхемы базы данных в среду языка четвертого поколения (см. раздел 3.3);

5) интеграция приложений: созданные приложения объединяются в единую среду и тестируются на совместимость; поскольку все приложения строились на основе общей глобальной модели данных, достигается высокая степень интеграции.

Silverrun имеет модульную структуру и состоит из следующих четырех модулей, каждый из которых является самостоятельным продуктом.

1. **Модуль построения моделей бизнес-процессов** в форме диаграмм потоков данных (BPM — Business Process Modeler) позволяет моделировать функционирование обследуемой организации или создаваемой системы. Диаграммы могут изображаться в различных нотациях, включая нотации Гейна — Сарсона и Йодана — Де Марко. Имеется также возможность создавать собственные нотации, в том числе добавлять в число изображаемых на схеме дескрипторов определенные пользователем поля.

2. **Модуль концептуального моделирования данных** (ERX — Entity-Relationship eXpert) обеспечивает построе-

ние моделей данных «сущность — связь», не привязанных к конкретной реализации. Этот модуль имеет встроенную экспертную систему, позволяющую создать корректную нормализованную модель данных посредством ответов на содержательные вопросы о взаимосвязи данных. Возможно автоматическое построение модели данных из описаний структур данных. Анализ функциональных зависимостей атрибутов дает возможность проверить соответствие модели требованиям третьей нормальной формы и обеспечить их выполнение. Проверенная модель передается в модуль RDM.

3. **Модуль реляционного моделирования** (RDM — Relational Data Modeler) позволяет создавать детализированные модели «сущность — связь», предназначенные для реализации в реляционной базе данных. В этом модуле документируются все объекты, связанные с построением базы данных: индексы, триггеры, хранимые процедуры и т.д. Гибкая изменяемая нотация и расширяемость репозитория позволяют работать по любой методологии. Возможность создавать подсхемы соответствует подходу ANSI-SPARC к представлению схемы базы данных. На языке подсхем моделируются как узлы распределенной обработки, так и пользовательские представления. Этот модуль обеспечивает проектирование и полное документирование реляционных баз данных.

4. **Менеджер репозитория рабочей группы** (WRM — Workgroup Repository Manager) применяется как словарь данных для хранения общей для всех моделей информации, а также обеспечивает интеграцию модулей Silverrun в единую среду проектирования.

Высокая гибкость и разнообразие изобразительных средств построения моделей в Silverrun не позволяют обеспечить жесткий взаимный контроль между компонентами различных моделей.

Для автоматической генерации схем баз данных в Silverrun существуют интерфейсы к наиболее распространенным СУБД (Oracle, SQL Server, Informix, DB2 и др.). Для передачи данных в средства разработки приложений имеются интерфейсы с языками четвертого поколения (PowerBuilder, SQL Windows, Uniface, NewEra, Delphi).

ER/Studio

ER/Studio предназначен для проектирования, создания и сопровождения баз данных [20]. ER/Studio предоставляет

мощные средства проектирования, которые обеспечивают интегрированный подход к решению задач, возникающих у специалистов, обслуживающих корпоративные базы данных.

Администраторы данных получают мощные средства логического моделирования, включающие стандартизацию и повторное использование элементов данных; администраторы баз данных — эффективную поддержку всех этапов жизненного цикла данных для разнообразных СУБД, включая автоматизированную генерацию кода, обратное проектирование («reverse engineering») и внесение изменений; разработчики прикладных систем получают инструмент разработки приложений, основанный на использовании языка UML; конечные пользователи благодаря наличию мощных средств ER/Studio для публикации документации в WWW могут анализировать использование данных в работе предприятия. Далее перечисляются некоторые возможности, имеющиеся в ER/Studio, а именно:

а) *многоуровневые средства проектирования* позволяют осуществлять разработку инфологических моделей. При этом результаты разработки могут быть трансформированы в любое количество даталогических моделей, которые поддерживают единые или различные системные каталоги СУБД;

б) *анализ использования* позволяет отобразить логические сущности и атрибуты на их реализацию на уровне даталогической модели. Соответствующий интерфейс дает возможность также отобразить распределение сущностей по подмоделям в рамках единой модели;

в) *автоматизированные преобразования* существенно упрощают построение даталогических моделей на базе логического представления. При этом ER/Studio проверяет выполнение условий нормализации и синтаксис в соответствии с правилами выбранной целевой СУБД. Для применения на уровне даталогической модели могут быть определены соглашения об именах и параметры, связанные с использованием памяти;

г) *денормализующие преобразования*, реализуемые Мастером денормализации, оптимизируют даталогическую модель базы данных, полученную на основе инфологической. Используется набор популярных стратегий денормализации, например, горизонтальные и вертикальные расщепления, расширяющие и сворачивающие преобразования таблиц, отображения колонок и другие. Что особенно важно, денормализующие преобразования в ER/Studio сохраняют связь с инфологической моделью, и конкретные логические объекты

остаются привязанными к тем даталогическим денормализованным объектам, к которым они относятся;

д) *мощные графические средства и функции автоматического размещения объектов*. Будучи средством визуального проектирования, ER/Studio имеет совершенные средства для ясного представления сложных моделей и осуществления легкой навигации. ER/Studio имеет набор эффективных утилит по размещению объектов, позволяющих одним щелчком преобразовывать диаграммы по определенному принципу, например иерархическому, в соответствии с которым «родители» следуют за «детьми», а те — за «внуками», в соответствии с распространением ключей;

е) *многоуровневые подмодели позволяют сохранять управляемость модели* при ее росте. ER/Studio позволяет легко создавать подмодели для выделения важных участков из больших и сложных моделей. При этом автоматически поддерживаются все связи и зависимости между подмоделями и всей диаграммой в целом;

ж) *нотации IDEF1X, IE и фильтруемая IE*. ER/Studio поддерживает три нотации моделей данных: IDEF1X, Information Engineering, а также фильтруемую IE нотацию, которая скрывает все внешние ключи. Можно использовать настройки по умолчанию для всех трех видов диаграмм или оперативно изменить их и выбрать такую нотацию, которая лучше всего подойдет для представления информации;

з) *многомерное моделирование*. В состав ER/Studio входят специальные средства, с помощью которых можно осуществить моделирование баз данных для систем принятия решений, в том числе построить модели хранилищ данных. Эти средства позволяют легко конструировать, публиковать и реализовывать базы данных, в которых используются сложные схемы типа «звезда» или «снежинка», а также обеспечивают двусторонний обмен многомерными метаданными.

ER/Studio поддерживает многочисленные версии таких СУБД, как DB2, Informix, InterBase, Access, SQL Server, Visual FoxPro, MySQL, Teradata, Oracle, Sybase Adaptive Server.

Design/IDEF

Design/IDEF является комплексным средством автоматизированного проектирования систем и объединяет несколько методологий, предназначенных для построения моделей определенного уровня [6]. С помощью Design/IDEF можно

автоматизировать многие этапы проектирования сложных систем различного назначения: формулировка требований и целей проектирования, разработка спецификаций, определение компонентов и взаимодействий между ними, составление документации, проверка полноты и непротиворечивости проекта. В Design/IDEF реализована методология структурного проектирования и анализа сложных систем *IDEF0/SADT* [2]. Design/IDEF строит иерархические модели сложных систем посредством декомпозиции; поддерживает коллективную разработку IDEF-модели, позволяя в любой момент объединять различные подмодели; создает словарь данных для хранения информации о функциях и структурах данных проекта; формирует отчет, поддерживающий процесс разработки и анализа моделей.

Кроме IDEF0/SADT, Design/IDEF поддерживает методологию моделирования динамики систем *IDEF/CPN*, основанную на «цветных» или «раскрашенных» сетях Петри [14], и методологию моделирования данных *IDEF1X* [2], основанную на ER-диаграммах. Процесс построения ER-диаграмм сводится к описанию реляционной модели данных, в котором каждой сущности на диаграмме соответствует таблица базы данных.

ERwin

ERwin входит в состав пакета инструментальных средств AllFusion Modeling Suite [16], поддерживающего все этапы разработки информационных систем, и используется при моделировании и создании баз данных произвольной сложности на основе ER-диаграмм. ERwin является популярным средством моделирования данных, благодаря поддержке широкого спектра СУБД самых различных классов от SQL-серверов (Oracle, Informix, Sybase Adaptive Server, SQL Server, Progress, DB2, SQLBase, Ingress, Rdb и др.) до «настольных» СУБД Clipper, dBase, FoxPro, MS Access, Paradox.

Инфологическая модель представляется в виде ER-диаграмм, отражающих основные сущности предметной области и связи между ними. Дополнительно определяются атрибуты сущностей, характеристики связей, индексы и правила, описывающие ограничения и закономерности предметной области. После создания ER-диаграммы ERwin автоматически генерирует SQL-операторы для создания таблиц, индексов и других объектов базы данных. По заданным разработчи-

ком правилам формируются стандартные триггеры БД для поддержки целостности данных; для сложных правил можно создавать собственные триггеры, используя библиотеку шаблонов.

Для изображения моделей можно использовать нотации, принятые в методологиях IDEF1X и IE.

ERwin может осуществлять обратное проектирование для существующих БД, генерируя ER-диаграммы по SQL-текстам. Таким образом, он полностью поддерживает технологию прямого и обратного построения, включающую следующие этапы:

1) импорт существующей БД с SQL-сервера;
2) автоматическая генерация модели БД;
3) модификация модели;
4) автоматическая генерация новой схемы и создание БД на том же самом или любом другом SQL-сервере.

Более конкретно возможности по проектированию БД, имеющиеся в ERwin, рассматриваются в следующих разделах.

Методологии создания ИЛМ

Различные методологии создания ИЛМ, используемые CASE-средствами, предлагают разработчику БД несколько иной набор условных обозначений для изображения диаграмм ER-типа, чем изложенный в разде. 6.3. Например, в программе ERwin [8, 16] реализуется *методология IDEF1X*, в которой сущности делятся на независимые и зависимые.

Независимая сущность — это сущность, каждый экземпляр которой может быть идентифицирован без учета его подчиненности другим сущностям. Примерами независимых сущностей являются ОТДЕЛ и СЛУЖАЩИЙ, если всем экземплярам каждой из этих сущностей присвоить уникальные номера, которые будут значениями атрибутов DepId (номер отдела) и EmpId (табельный номер служащего). В этом случае, несмотря на то, что каждый служащий «подчинен» отделу, он идентифицируется своим номером независимо от отдела, в котором работает. Аналогичным образом каждый отдел идентифицируется независимо от служащих, которые в нем работают. Независимая сущность изображается в виде прямоугольного блока, внутри которого указан список атрибутов. Атрибуты, входящие в ключ сущности, размещаются в начале списка и отделяются от других атрибутов горизонтальной чертой (рис. 6.15, *а*).

Рис. 6.15. **Графическое обозначение независимых (*а*) и зависимой (*б*) сущностей**

Зависимая сущность — это сущность, однозначная идентификация экземпляра которой зависит от его подчиненности другой сущности. Примером зависимой сущности служит сущность РЕБЕНОК, если для обозначения каждого экземпляра использовать имя ребенка, родителем которого является служащий. Поскольку у разных служащих могут быть дети с одинаковыми именами, то однозначная идентификация ребенка зависит от его «подчиненности» своему родителю — служащему. Зависимая сущность изображается в виде блока с закругленными углами (рис. 6.15, *б*).

При рассмотрении связи двух сущностей подчиненная сущность называется сущностью-потомком, а подчиняющая сущность — сущностью-родителем.

Связь сущностей характеризуется идентификацией и степенью.

Идентифицирующая связь, обозначаемая сплошной линией, соединяет сущность-родителя с зависимой сущностью-потомком (рис. 6.16, *а*) и задает на диаграмме обязательный класс принадлежности для сущности-потомка и степень связи $1 : N$ (или $1 : 1$).

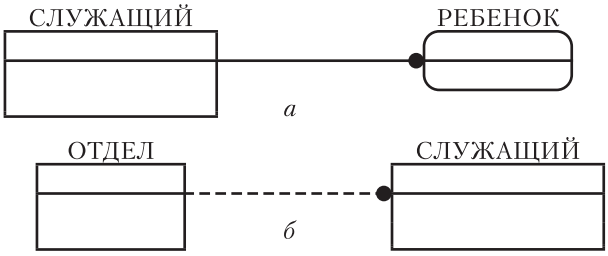


Рис. 6.16. **Графическое обозначение идентифицирующей (*а*) и неидентифицирующей (*б*) связей между сущностями**

Неидентифицирующая связь, обозначаемая штриховой линией, соединяет сущность-родителя с независимой сущностью-потомком и представляет степень связи 1 : *N* или 1 : 1 (рис. 6.16 *б*).

Методология IDEF1X позволяет представить изображаемые на ER-диаграммах классы принадлежности и степени связи (1 : 1, 1 : *N*, *N* : 1, *N* : *M*) с помощью идентифицирующей и неидентифицирующей связей и связи «многие ко многим» (рис. 6.17).

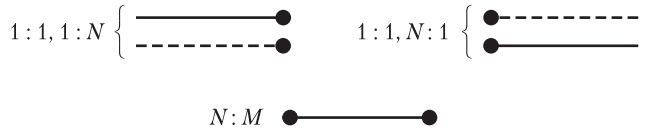


Рис. 6.17. **Представление степеней связи**

Степень связи 1 : *N* со стороны «многие» характеризуется мощностью (cardinality), которая обозначает количество экземпляров сущности-потомка (СП), существующих для каждого экземпляра сущности-родителя (СР). Мощность позволяет задать обязательный или необязательный класс принадлежности для сущности-потомка и может принимать значения, указанные в табл. 6.2.

Для сущности-родителя класс принадлежности графически не обозначается и в случае неидентифицирующей связи не является существенным для определения соответствующей ДЛМ.

6.8.2. Проектирование баз данных с использованием ERwin

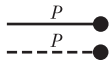
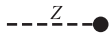
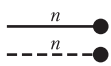
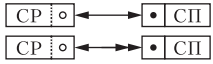
ERwin предназначен для построения ИЛМ с использованием методологий IDEF1X или IE и автоматической генерации соответствующей ДЛМ с учетом особенностей выбранной СУБД.

Таблица 6.2

Представление степеней связи и классов принадлежности

Мощность	Графическое обозначение связи (IDEF1X)	Степень связи	Класс принадлежности для СП и ER-диаграмма	Пояснение
N		1 : *N*		Одному экземпляру СР соответствует 0, 1 или много экземпляров СП

Окончание табл. 6.2

Мощность	Графическое обозначение связи (IDEF1X)	Степень связи	Класс принадлежности для СП и ER-диаграмма	Пояснение
P		1 : *N*		Одному экземпляру СР соответствует 1 или много экземпляров СП
Z		1 : 1		Одному экземпляру СР соответствует 0 или 1 экземпляр СП
n		1 : 1 1 : *n*		Одному экземпляру СР соответствует ровно *n* экземпляров СП (например, *n* = 1 или *n* = 5)

Результатом генерации ДЛМ является схема БД, представленная на языке SQL, или формирующая базу данных программа (например, на языке Visual Basic) и созданные таблицы, входящие в БД.

Для обозначения моделей данных ERwin использует терминологию, отличную от рассмотренной ранее: ИЛМ именуется логической (Logical) моделью, а ДЛМ — физической (Physical) моделью.

Создание БД с помощью ERwin начинается с построения логической модели. После описания логической модели проектировщик выбирает необходимую СУБД, и ERwin автоматически создает соответствующую физическую модель. На основе физической модели ERwin может сгенерировать схему БД и сформировать таблицы, образующие БД.

Этот процесс называется прямым проектированием (Forward Engineering) и обеспечивает масштабируемость: создав одну логическую модель, можно сгенерировать физические модели для любой СУБД, которая поддерживается в ERwin.

Кроме того, ERwin способен для существующей БД воссоздать физическую и логическую модели, т.е. обеспечить обратное проектирование (Reverse Engineering). На основе полученной логической модели можно сгенерировать физическую модель для другой СУБД и затем сформировать новую БД. Следовательно, ERwin позволяет решить задачу по переносу структуры БД с одной СУБД на другую.

После запуска появляется окно программы (рис. 6.18) со строкой главного меню, панелью инструментов и рабо-

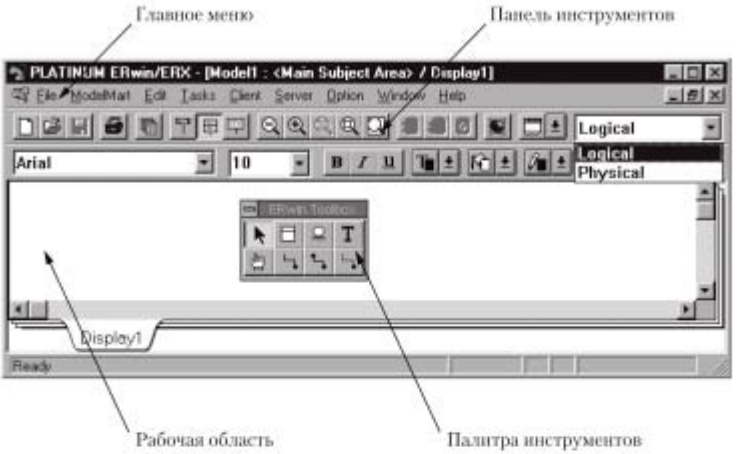


Рис. 6.18. **Окно программы**

чей областью, в которой находится палитра инструментов с кнопками.

Вид палитры инструментов (ERwin Toolbox) зависит от выбора логической или физической модели, осуществляемого с помощью списка, расположенного в правой части панели инструментов (см. рис. 6.18), а также от указанной разработчиком нотации. Для логической модели в зависимости от выбранной методологии IDEF1X или IE палитра инструментов имеет кнопки, назначение которых указано в табл. 6.3.

Рассмотрим в качестве предметной области предприятие, в структуре которого имеются отделы, и спроектируем БД для хранения сведений о служащих, работающих в отделах, и их детях. Описание сущностей и связей между ними представлено на рис. 6.19.

Таблица 6.3

Кнопки палитры инструментов

Кнопка		Назначение	Кнопка		Назначение
IDEF1X	IE		IDEF1X	IE	
		Указатель элемента модели (элементами модели являются сущности и связи)			Перемещение атрибутов внутри сущностей или между сущностями (способом drag & drop)
		Создание сущности			Создание идентифицирующей связи

Окончание табл. 6.3

Кнопка		Назначение	Кнопка		Назначение
IDEF1X	IE		IDEF1X	IE	
		Создание категории, или категориальной связи (используется для описания обобщенных объектов (сущностей))			Создание связи $M:N$ (многие ко многим)
T	T	Добавление текстового блока в модель			Добавление неидентифицирующей связи

На ER-диаграмме атрибут ChiName (Имя ребенка) подчеркнут штриховой линией, чтобы указать на то, что по имени можно идентифицировать ребенка, только «подчинив» его служащему-родителю, т.е. ребенок является зависимой сущностью по отношению к служащему. (Сделать сущность РЕБЕНОК независимой можно, если пронумеровать всех детей и использовать их номера для однозначной идентификации.)

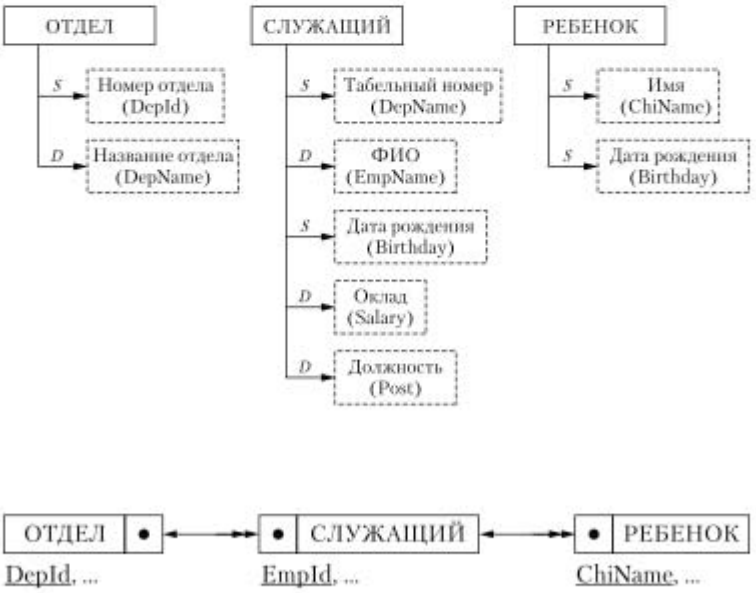


Рис. 6.19. **Описание объектов и связей между ними**

Связь объектов, показанную на ER-диаграмме, необходимо представить в соответствии с методологией IDEF1X (см. табл. 6.2), как показано на рис. 6.20.

Рис. 6.20. **Логическая модель**

Полученная диаграмма описывается средствами ERwin и помещается в файл с расширением ER1. После выбора СУБД формируется физическая модель БД как совокупность взаимосвязанных таблиц. Для последующего использования БД удобнее, чтобы имена таблиц и атрибутов записывались латинскими буквами, поскольку не все СУБД допускают работу с кириллицей. В табл. 6.4 приведено соответствие между именами логической и физической моделей и указаны типы атрибутов.

Таблица 6.4

Соответствие между именами логической и физической моделей

Имя		Имя		Тип	
сущности	**таблицы**	**атрибута**	**столбца**	**атрибута**	**столбца**
ОТДЕЛ	Department	Номер отдела	DepId	Числовой	Number
		Название отдела	DepName	Строковый	String
СЛУЖАЩИЙ	Employee	Табельный номер	EmpId	Числовой	Number
		ФИО	EmpName	Строковый	String
		Дата рождения	Birthday	Дата	Datetime
		Оклад	Salary	Числовой	Number
		Должность	Post	Строковый	String
РЕБЕНОК	Children	Имя	ChiName	Строковый	String
		Дата рождения	Birthday	Дата	Datetime

Чтобы описать сущности, входящие в логическую модель, показанную на рис. 6.20, нужно на панели инструментов задать режим создания логической модели (Logical) и командой File | New создать новую модель, выбрав в поя-

вившемся окне шаблон Blank Diagram и нажав кнопку OK. Для использования нотации, соответствующей методологии IDEF1X, нужно командой Option | Preferences активизировать окно Preferences, в котором на закладке Methodology выбрать нужную нотацию.

Далее на палитре инструментов нажимается кнопка создания сущности и в рабочем поле диаграммы щелчком мыши размещаются три сущности. Курсором мыши указывается первая сущность в рабочем поле диаграммы и из контекстного меню выбирается команда Entity Editor. В области Name диалогового окна Entity Editor набирается имя сущности Department, которое будет использоваться в модели вместо имени ОТДЕЛ. На закладке Definition в одноименной области следует набрать определение сущности (рис. 6.21, *а*).

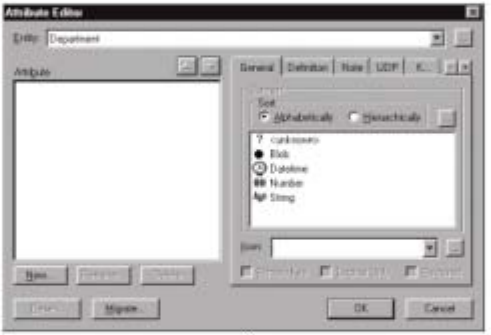
Для описания атрибутов сущности Department из контекстного меню этой сущности выбирается команда Attribute Editor, в одноименном диалоговом окне (рис. 6.21, *б*) нажимается кнопка New и в появившемся диалоговом окне New Attribute (рис. 6.21, *в*) указывается имя атрибута, имя соответствующего столбца в таблице БД и тип данных, хранящихся в столбце (домен): в области Attribute Name — *Номер отдела*, в области Column Name — *DepId*, в области Domain — числовой тип *Number*. Описанный атрибут является ключом сущности, поэтому для него после возврата в диалоговое окно Attribute Editor на закладке General следует установить признак Primary Key.

Аналогичным образом согласно табл. 6.4 описывается атрибут *Название отдела*, но без включения его в состав ключа сущности.

Для описания сущностей СЛУЖАЩИЙ и РЕБЕНОК следует выполнить рассмотренные выше действия применительно к этим сущностям согласно табл. 6.4.

Чтобы задать связи между сущностями (см. рис. 6.20), нужно соединить сущности ОТДЕЛ и СЛУЖАЩИЙ неидентифицирующей связью, нажав на палитре инструментов кнопку со штриховой линией, а затем последовательно щелкнув мышью по сущностям ОТДЕЛ и СЛУЖАЩИЙ. Таким образом будет установлена связь «один ко многим». При этом ключ сущности-родителя появится среди атрибутов сущности-потомка и будет помечен как внешний ключ (FK). Далее необходимо щелкнуть правой кнопкой мыши по связи и в контекстном меню выбрать команду Relationship Editor, чтобы установить такие характеристики связи, как имя связи

а



б

в

Рис. 6.21. **Описание сущности ОТДЕЛ (*а*) и ее атрибутов (*б*, *в*)**

(Verb Phrase) и мощность (Cardinality) в диалоговом окне Relationship Editor (рис. 6.22).

В области Parent-to-Child этого окна задается имя связи со стороны сущности-родителя (*состоит из*), а в области Child-to-Parent — со стороны сущности-потомка (*работает в*). Мощность связи устанавливается выбором радиокнопки One or More (P) в области Cardinality, а обязательный класс принадлежности для сущности-потомка СЛУЖАЩИЙ — выбором радиокнопки No Nulls в области Relationship Type.

Рис. 6.22. **Окно редактора связей**

Аналогичным способом, но идентифицирующей связью, соединяются сущности СЛУЖАЩИЙ и РЕБЕНОК: на палитре инструментов нажимается кнопка со сплошной линией и последовательно делаются щелчки мышью по сущностям СЛУЖАЩИЙ и РЕБЕНОК. В результате будет установлена связь «один ко многим», сущность РЕБЕНОК будет изображена как зависимая, а ключ сущности-родителя появится среди атрибутов сущности-потомка, образующих ее ключ, и будет помечен как внешний ключ (FK). После установки имени (*имеет/принадлежит*) и мощности (*P*) этой связи ER — диаграмма примет вид, показанный на рис. 6.23.

Для полученной логической модели можно сформировать соответствующую физическую модель, раскрыв на панели инструментов расположенный справа список (см. рис. 6.18)

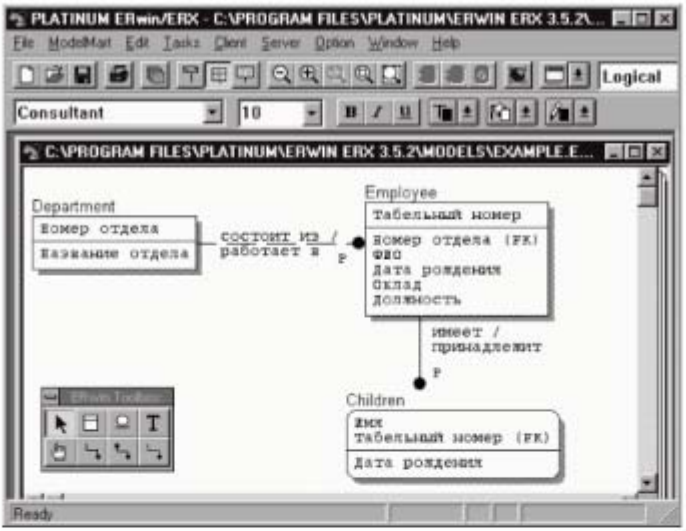


Рис. 6.23. **Логическая модель, созданная в ERwin**

и выбрав из него тип модели Physical. Чтобы адаптировать спроектированную таким образом базу данных для СУБД Oracle, нужно в главном меню выбрать команду Server | Target Server и в появившемся диалоговом окне (рис. 6.24) из списка серверных СУБД (Target SQL DBMS) выбрать радиокнопку ORACLE. В результате получится физическая модель, показанная на рис. 6.25.

После этого можно создать БД для выбранной СУБД командой главного меню Tasks | Forward Engineering/Schema Generation. Эта команда активизирует диалоговое окно, в котором представлены режимы генерации объектов БД — таблиц, столбцов, индексов и т.д. Нажав кнопку Preview, можно вызвать окно, в котором отображаются операторы языка SQL для создания БД и задающие схему БД. Создание БД запускается нажатием кнопки Generate в этом окне. Если установлена связь с СУБД, то протокол создания БД отображается в окне Generate Database Schema.

Полученная физическая модель легко преобразуется в формат любой СУБД, предусмотренной в ERwin, что позволяет построить распределенную базу данных, доступ к отдельным частям которой будет обеспечиваться соответствующими СУБД.

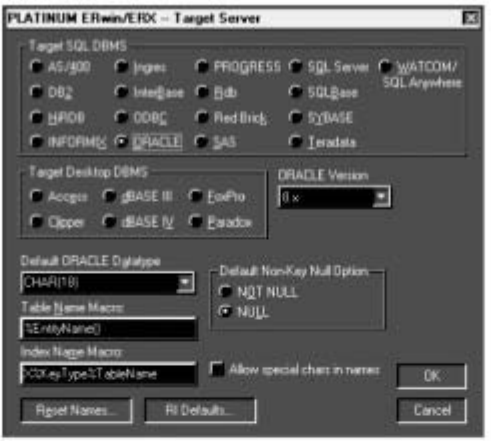


Рис. 6.24. **Окно выбора СУБД**

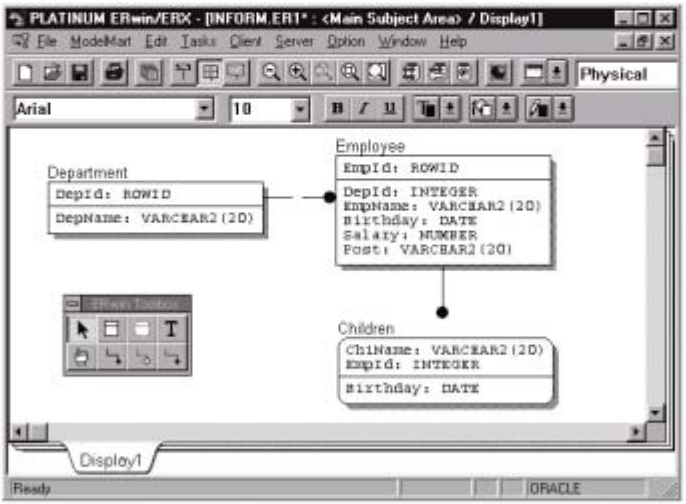


Рис. 6.25. **Физическая модель для СУБД Oracle**

Глоссарий

Атрибут — свойство, описывающее определенный аспект объекта, значение которого следует зафиксировать в описании *предметной области*

База данных — поименованная совокупность взаимосвязанных данных, управляемых специальной системой, называемой *системой управления базами данных* (СУБД)

Банк данных — система, состоящая из баз данных, программных, технических, языковых, организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных

Блокировка — запрет выполнения некоторых операций над данными (например, обновления или добавления), если с ними работает другой пользователь

Браузер — специальная программа для просмотра гипертекстовых документов

Гипертекст — размеченный текст, содержащий ссылки на внешние информационные ресурсы

Дескриптор — описатель документа или его фрагмента. Дескрипторы предусмотрены, в частности, в языках HTML и XML и используются для разметки документа

Журнал — системная структура или специальный файл, которые СУБД использует для фиксации хода выполнения *транзакций*; содержит сведения обо всех изменениях, производимых в базе данных

Журнал транзакций — *См. Журнал*

Индекс — системная структура или специальный файл, создаваемые для повышения скорости выполнения запросов; индекс формируется из значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы и позволяет быстро находить нужную строку таблицы по заданному значению

Интернет-портал — *сайт*, предоставляющий пользователю Интернета возможность дальнейшего получения информации

с других сайтов за счет большого количества внешних ссылок, указывающих на другие ресурсы

Интернет-приложение — *См. Web-приложение*

Интранет (intranet) — компьютерные сети, использующие *Web-технологию* для доступа к данным

Ключ внешний — атрибут (столбец), или набор атрибутов, подчиненной таблицы, который в главной таблице является *первичным ключом*

Ключ возможный — атрибут (столбец) или набор атрибутов, который может быть использован для данного отношения (таблицы) в качестве *первичного ключа*

Ключ индексный — атрибут (столбец) или набор атрибутов, определенный в отношении (таблице) и используемый для формирования *индекса*

Ключ первичный — атрибут отношения (столбец таблицы), или набор из минимального числа атрибутов, который однозначно идентифицирует конкретный кортеж в отношении (конкретную строку в таблице)

Компиляция — преобразование программы из описания на входном языке (языке программирования) в ее представление на выходном языке (в машинных командах)

Метаданные — информация, описывающая данные, хранящиеся в базе данных

Метаинформация — информация, описывающая базу данных, а также другие части банка данных

Нормализация — разбиение отношения на два или более отношений, обладающих лучшими свойствами при добавлении, обновлении и удалении данных

Предметная область — часть реального мира, представляющая интерес для данного исследования или использования

Привилегия — право пользователей на выполнение определенных операций над объектами базы данных

Приложение — прикладная компьютерная программа, предназначенная для решения определенной практической задачи

Репликация — технология, обеспечивающая поддержку копий базы данных или ее фрагментов в нескольких узлах компьютерной сети

Сайт — совокупность *Web-страниц*, доступных в Интернете; Web-страницы сайта объединены общим адресом, а также обычно темой, логической структурой, оформлением или авторством

Свойство — характеристика сущности

Связь — ассоциативное отношение между *сущностями*, при котором каждый экземпляр одной сущности соединен с некоторым (в том числе нулевым) количеством экземпляров другой сущности

Семантическая сеть — способ представления знаний или смысла текста в виде ориентированного графа, в котором вершины соответствуют понятиям, объектам, действиям, ситуациям или сложным отношениям, дуги — свойствам и элементарным отношениям

Сериализуемость — критерий корректности управления одновременным доступом к данным, который требует, чтобы результат множества одновременно выполняемых транзакций был эквивалентен результату от их последовательного выполнения при каком-либо упорядочении

Система баз данных — совокупность программного обеспечения, данных и аппаратного обеспечения компьютеров, которая реализует набор приложений и моделей данных и использует СУБД и прикладное программное обеспечение для создания конкретной информационной системы

Система управления базами данных — совокупность специальных языковых и программных средств, облегчающих пользователям выполнение всех операций, связанных с организацией хранения данных, их корректировкой и доступом к ним

Сущность — отдельный класс объектов предметной области, который должен быть представлен в базе данных

Таблица кодировки символов — таблица, устанавливающая соответствие между символами и их числовыми кодами. Наиболее распространенными кодами являются ASCII (American Standard Code for Information Interchange — американский стандартный код для обмена информацией), EBCDIC (Extended Binary Coded Decimal Interchange Code - расширенный двоично-десятичный код обмена информацией) и Unicode (Юникод, или Уникод, — универсальная система кодирования). В кодах EBCDIC и ASCII символы кодируются однобайтовыми целыми числами, а в коде Unicode — одно-, двух- или четырехбайтовыми целыми числами

Транзакция — последовательность действий над базой данных, переводящая базу данных из одного непротиворечивого (целостного) состояния в другое непротиворечивое состояние

Трансляция — *См. Компиляция*

Web-портал — *См. Интернет-портал*

Web-приложение — специальное расширение *Web-сервера*, предназначенное для решения конкретной задачи, например, получения информации из базы данных

Web-сервер — программа, находящаяся на *WWW-сервере* и обеспечивающая доступ к документам

Web-сервис — идентифицируемая строкой URL программная система, чьи доступные интерфейсы определены и описаны языком XML; описание этой программной системы может быть найдено другими программными системами, которые могут взаимодействовать с ней согласно этому описанию посредством сообщений, основанных на XML

Web-страница — гипертекстовый документ, для просмотра которого используется *браузер*

Web-технологии — информационные технологии, обеспечивающие работу Всемирной паутины

WWW-сервер — серверный компьютер, предназначенный для размещения документов во Всемирной паутине

Литература

1. *Байдачный, С.,* SQL Server 2005: Новые возможности для разработчиков. / С. Байдачный, Д. Маленко, Ю. Лозинский. — М. : СОЛОН-Пресс, 2006. — 208 с.

2. *Вендров, А. М.* CASE-технологии. Современные методы и средства проектирования информационных систем / А. М. Вендров. — М. : Финансы и статистика, 1998. — 176 с.

3. *Вендров, А. М.* Проектирование программного обеспечения экономических информационных систем : учебник / А. М. Вендров. — 2-е изд. — М. : Финансы и статистика, 2005. — 544 с.

4. *Грофф, Дж. Р., Вайнберг, П. Н.* SQL: полное руководство / Дж. Р. Грофф, П. Н. Вайнберг. — 2-е изд. — К. : BHV, 2001. — 816 с.

5. *Джексон, Г.* Проектирование реляционных баз данных для использования с микроЭВМ / Г. Джексон. — М. : Мир, 1991. — 252 с.

6. *Диго, С. М.* Базы данных: проектирование и использование : учебник / С. М. Диго. — М. : Финансы и статистика, 2005. — 592 с.

7. *Избачков, Ю. С.* Информационные системы : учебник для вузов / Ю. С. Избачков, В. Н. Петров. — 2-е изд. — СПб. : Питер, 2006. — 656 с.

8. *Илюшечкин, В. М.* Лабораторный практикум по курсу «Базы данных» / В. М. Илюшечкин, Л. В. Илюшечкина. — Ч. 2. — М. : МИЭТ, 2004. — 104 с.

9. *Калянов, Г. Н.* CASE. Структурный системный анализ (автоматизация и применение) / Г. Н. Калянов. — М. : Лори, 1996. — 242 с.

10. *Карпова, Т. С.* Базы данных: модели, разработка, реализация / Т. С. Карпова. — СПб. : Питер, 2001. — 304 с.

11. *Ковязин, А. Н.* Мир InterBase. Архитектура, администрирование и разработка приложений баз данных в InterBase/Firebird/Yaffil / А. Н. Ковязин, С. М. Востриков. — 4-е изд. — М. : Кудиц-Образ, 2006. — 496 с.

12. *Колетски, П.* Oracle Designer. Настольная книга пользователя / П. Колетски, П. Дорси. — М. : Лори, 2000. — 596 с.

13. *Коннолли, Т.* Базы данных. Проектирование, реализация и сопровождение. Теория и практика / Т. Коннолли, К. Бегг. — 3-е изд. — М. : Изд. дом «Вильямс», 2003. — 1440 с.

14. *Котов, В. Е.* Сети Петри / В. Е. Котов. — М. : Наука, 1984. — 161 с.

15. *Кренке, Д.* Теория и практика построения баз данных / Д. Кренке. — 8-е изд. — СПб. : Питер, 2003. — 800 с.

16. *Маклаков, С. В.* Создание информационных систем с AllFusion Modeling Suite / С. В. Маклаков. — М. : ДИАЛОГ-МИФИ, 2003. — 432 с.

17. *Нейбург, Э. Дж.* Проектирование баз данных с помощью UML / Э. Дж. Нейбург, Р. А. Максимчук. — М. : Изд. дом «Вильямс», 2002. — 288 с.

18. *Риккарди, Г.* Системы баз данных. Теория и практика использования в Internet и среде Java / Г. Риккарди. — М. : Издательский дом «Вильямс», 2001. — 480 с.

19. *Фаронов, В. В.* Программирование баз данных в Delphi 7 : учебный курс / В. В. Фаронов. — СПб. : Питер, 2003. — 459 с.

20. ER/Studio User Guide. — San Francisco, CA : Embarcadero Technologies Inc., 2006. — 660 pp.

Наши книги можно приобрести:

Учебным заведениям и библиотекам:
в отделе по работе с вузами
тел.: (495) 744-00-12, e-mail: vuz@urait.ru

Частным лицам:
список магазинов смотрите на сайте urait.ru
в разделе «Частным лицам»

Магазинам и корпоративным клиентам:
в отделе продаж
тел.: (495) 744-00-12, e-mail: sales@urait.ru

Отзывы об издании присылайте в редакцию
e-mail: red@urait.ru

Новые издания и дополнительные материалы доступны
в электронной библиотечной системе «Юрайт»
biblio-online.ru

Учебное издание

Илюшечкин Владимир Михайлович

ОСНОВЫ ИСПОЛЬЗОВАНИЯ И ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ

Учебник для СПО

Формат 60×90 $^{1}/_{16}$.
Гарнитура «Petersburg». Печать цифровая.
Усл. печ. л. 13,31. Заказ №

ООО «Издательство Юрайт»
111123, г. Москва, ул. Плеханова, д. 4а.
Тел.: (495) 744-00-12. E-mail: izdat@urait.ru, www.urait.ru