

ФГБОУ ВПО
«Воронежский государственный
технический университет»

Т.И. Сергеева М.Ю. Сергеев

БАЗЫ ДАННЫХ: МОДЕЛИ ДАННЫХ,
ПРОЕКТИРОВАНИЕ, ЯЗЫК SQL

Утверждено Редакционно-издательским советом
университета в качестве учебного пособия

Воронеж 2012

УДК 681.3

Сергеева Т.И. Базы данных: модели данных, проектирование, язык SQL: учеб. пособие / Т.И. Сергеева, М.Ю. Сергеев. Воронеж: ФГБОУ ВПО «Воронежский государственный технический университет», 2012. 233 с.

В учебном пособии рассматриваются понятия баз данных и систем управления базами данных, модели данных, язык SQL, проектирование реляционных баз данных, физическая организация баз данных, защита информации в базах данных, построение распределенных баз данных.

Издание соответствует требованиям Федерального государственного образовательного стандарта высшего профессионального образования по направлению подготовки бакалавров 230100 «Информатика и вычислительная техника», профилю «Вычислительные машины, комплексы, системы и сети», дисциплине «Базы данных».

Учебное пособие подготовлено в электронном виде в текстовом редакторе Microsoft Word 2003 и содержится в файле BD.doc.

Табл. 5. Ил. 43. Библиогр.: 10 назв.

Научный редактор д-р техн. наук, проф. С.Л. Подвальный

Рецензенты: кафедра информатики и вычислительной техники Международного института компьютерных технологий
(зав. кафедрой канд. техн. наук,
доц. И.Н. Крючкова);
д-р техн. наук, проф. Т.М. Леденева

© Сергеева Т.И., Сергеев М.Ю., 2012

© Оформление. ФГБОУ ВПО «Воронежский государственный технический университет», 2012

ВВЕДЕНИЕ

Деятельность современного человека и функционирование преуспевающих предприятий и организаций невозможны без использования информационных систем, составной частью которых являются базы данных. Любая эффективная практическая деятельность в той или иной степени связана со сбором, хранением и обработкой все возрастающих объемов информации. Оперативная и эффективная обработка информации требует применения наиболее перспективных компьютерных технологий.

Освоение теоретических основ построения баз данных, перспективных моделей данных, различных средств проектирования баз данных, особенностей физической организации баз данных, средств защиты баз данных и обеспечения целостности и сохранности данных, особенностей построения и функционирования распределенных баз данных является целью данного пособия.

В первой главе учебного пособия изложены базовые понятия теории баз данных, назначение и основные компоненты системы баз данных, уровни представления баз данных, обзор современных систем управления базами данных.

Вторая глава содержит описание основных моделей данных. Рассматриваются классические модели данных и современные модели данных, которые стали активно внедряться в практику использования баз данных.

В третьей главе приведены теоретические основы построения реляционной модели данных, описание операций реляционной алгебры и стандартного языка работы с базами данных SQL.

Четвертая глава посвящена вопросам проектирования баз данных с использованием метода нормальных форм и метода сущность-связь, в том числе с использованием средств автоматизации проектирования.

В пятой главе описываются физическая организация базы данных, файловые структуры, используемые для хранения информации в базах данных, модели бесфайловой физической организации данных.

В шестой главе рассматриваются вопросы защиты информации в базах данных.

В седьмой главе кратко излагаются принципы построения распределенных баз данных.

Учебное пособие полностью соответствует типовой программе по дисциплине «Базы данных» и предназначено для студентов очной полной и сокращенной форм обучения.

1. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

1.1. Информационные системы и банк данных

Использование банков *данных* и других информационных систем становится неотъемлемой составляющей деловой деятельности современного человека и функционирования преуспевающей организации. В связи с тем, что объемы информации, подлежащей хранению и обработке, растут высокими темпами, производители программного обеспечения вынуждены разрабатывать новые гибкие подходы к управлению большими объемами данных. В таких условиях большую актуальность приобретает освоение принципов построения современных информационных систем и эффективного применения соответствующих технологий и программных продуктов.

В основе решения многих задач лежит обработка информации. Для облегчения обработки информации создаются информационные системы (ИС). Автоматизированными называются ИС, в которых применяются технические средства, в частности ЭВМ, для хранения и обработки информации. Большинство существующих ИС являются автоматизированными, поэтому для краткости просто будем называть их ИС.

В широком понимании ИС – это любая система обработки информации.

По области применения ИС разделяют на системы, используемые в производстве, образовании, здравоохранении, науке, военном деле, социальной сфере, торговле и других отраслях.

По целевому назначению ИС можно условно разделить на следующие основные категории: управляющие, информационно-справочные, поддержки принятия решений.

Более узко ИС трактуют как совокупность аппаратно-программных средств, предназначенных для решения некоторой прикладной задачи. В организации, например, могут существовать информационные системы, выполняющие следующие задачи: учет кадров, учет материально-технических средств, расчет с поставщиками и заказчиками, бухгалтерский учет и т.д.

Банк данных является разновидностью ИС. Банк данных осуществляет **централизованное** хранение и накопление обрабатываемой информации, организованной в одну или несколько баз данных.

Автоматизированный банк данных определяют как систему информационных, математических, программных, языковых, организационных и технических средств, предназначенных для централизованного накопления и коллективного многоаспектного использования данных с целью получения необходимой информации.

1.2. Назначение и основные компоненты банка данных

Банк данных (БНД) в общем случае состоит из следующих компонентов: *базы данных* (или нескольких баз данных), системы управления базами данных (СУБД), словаря данных, администратора базы данных, вычислительной системы и обслуживающего персонала.

Рассмотрим вкратце названные компоненты и некоторые связанные с ними понятия.

База данных (БД) представляет собой совокупность специальным образом организованных данных, хранимых в памяти вычислительной системы и отображающих состояние объектов и их взаимосвязей в рассматриваемой предметной области. Данные, организованные в базу, обладают такой минимальной избыточностью, которая допускает их многоцелевое использование.

Данные, хранимые в базе, имеют определенную логическую структуру, которую называют **моделью представления данных**. **Модель данных** – средство абстракции, позволяющее видеть обобщенную структуру данных, хранимых в базе данных, а не их конкретные значения.

К основным моделям представления данных (моделям данных) относятся следующие: иерархическая, сетевая, реляционная, постреляционная, объектно-реляционная, многомерная и объектно-ориентированная.

Система управления базами данных (СУБД) – это комплекс языковых, математических и программных средств, предназначенных для централизованного создания, ведения и совместного использования БД многими пользователями. Таким образом, СУБД – это совокупность программных средств, обеспечивающих централизованное управление базой данных.

Обычно СУБД различают по используемой модели данных. Так СУБД, основанные на использовании реляционной модели данных, называют реляционными СУБД. Количество современных систем управления базами данных исчисляется тысячами. С СУБД связано понятие приложения.

Приложение – программа или комплекс программ, обеспечивающих автоматизацию обработки информации для прикладной задачи. Нами рассматриваются приложения, использующие базы данных. Приложения могут создаваться в среде и вне среды СУБД. Вне среды СУБД приложения создаются с помощью системы программирования, использующей средства доступа к базе данных. Примерами таких систем являются: Delphi или C++ Builder. Приложения, разработанные в среде СУБД,

часто называют приложениями СУБД, а приложения, разработанные вне СУБД, - внешними приложениями.

Словарь данных (СД) представляет собой подсистему БД, предназначенную для централизованного хранения информации о структурах данных, взаимосвязях файлов базы данных друг с другом, типах данных и форматах их представления, принадлежности данных пользователям, кодах защиты и ограничения доступа и т.д.

Функционально словарь данных присутствует во всех банках данных, но не всегда выполняющий эти функции компонент имеет именно такое название. Чаще всего функции словаря данных выполняются СУБД и вызываются из основного меню системы или реализуются с помощью ее утилит.

Администратор базы данных (АБД) – это лицо или группа лиц, отвечающих за выработку требований к базе данных, проектирование базы данных, ее создание, эффективное использование и сопровождение.

В процессе проектирования базы данных АБД осуществляет анализ предметной области, проектирование структуры БД, организационно-методическую работу по проектированию БД, задание ограничений целостности при описании структуры БД и процедур обработки БД.

В процессе создания базы данных АБД осуществляет первоначальную загрузку и ведение БД.

В процессе эксплуатации АБД обычно следит за функционированием информационной системы, обеспечивает защиту данных от несанкционированного доступа, контролирует избыточность, непротиворечивость, сохранность и достоверность хранимой в базе данных информации, обеспечивает восстановление БД, анализирует обращения пользователей, анализирует эффективность функционирования БД, работает с конечными пользователями, занимается модернизацией программных средств и развитием БД.

Для однопользовательских информационных систем функции АБД обычно возлагаются на лиц, непосредственно работающих с приложением базы данных.

В вычислительной сети АБД, как правило, взаимодействует с администратором сети. В обязанности последнего входят контроль за функционированием аппаратно-программных средств сети, реконфигурация сети, восстановление программного обеспечения после сбоев и отказов оборудования, профилактические мероприятия и обеспечение разграничения доступа.

Вычислительная система (ВС) представляет собой совокупность взаимосвязанных и согласованно действующих ЭВМ или процессоров и других устройств, обеспечивающих автоматизацию процессов приема, обработки и выдачи информации потребителям. Поскольку основными функциями банка данных являются хранение и обработка данных, то используемая вычислительная система, наряду с приемлемой мощностью центральных процессоров (ЦП), должна иметь достаточный объем оперативной и внешней памяти прямого доступа.

Обслуживающий персонал выполняет функции поддержания технических и программных средств в работоспособном состоянии. Он проводит профилактические, регламентные, восстановительные и другие работы по планам, а также по мере необходимости.

Как любой программно-организационно-технический комплекс, банк данных существует во времени и в пространстве. Он имеет определенные стадии своего развития: проектирование, реализацию, эксплуатацию, модернизацию и развитие, полную реорганизацию. На каждом этапе своего существования с банком данных связаны разные **категории пользователей**.

Конечные пользователи. Это основная категория пользователей, в интересах которых и создается банк данных. Главный принцип состоит в том, что от конечных пользователей не должно требоваться каких-либо специальных знаний в области вычислительной техники и языковых средств.

Администраторы банка данных. Это группа пользователей, которая на начальной стадии разработки банка данных отвечает за его оптимальную организацию с точки зрения одновременной работы множества конечных пользователей, на стадии эксплуатации отвечает за корректность работы данного банка информации в многопользовательском режиме. На стадии развития и реорганизации эта группа пользователей отвечает за возможность корректной реорганизации банка без изменения или прекращения его текущей эксплуатации.

Разработчики и администраторы приложений. Эта группа пользователей функционирует во время проектирования, создания и реорганизации банка данных. Администраторы приложений координируют работу разработчиков при создании конкретного приложения или группы приложений, объединенных в функциональную подсистему.

Не в каждом банке данных могут быть выделены все типы пользователей. При разработке информационных систем с использованием настольных СУБД администратор банка данных, администратор приложений и разработчик часто существовали в одном лице. Однако при построении современных сложных корпоративных баз данных, которые используются для автоматизации всех или большей части бизнес-процессов в крупной фирме или корпорации, могут существовать и группы администраторов приложений, и отделы разработчиков. Наиболее сложные обязанности возложены на группу администратора БД.

1.3. Архитектура базы данных. Физическая и логическая независимость данных

База данных – именованная совокупность данных, отражающая состояние объектов и их отношений в рассматриваемой предметной области. База данных – совокупность связанных данных, описание, хранение и манипулирование которыми осуществляется по общим правилам.

Цель любой информационной системы – обработка данных об объектах реального мира. В широком смысле слова база данных – это совокупность сведений о конкретных объектах реального мира в какой–либо предметной области.

Данные, организованные в базу, имеют минимальную избыточность и могут быть использованы оптимальным образом при решении различных задач. Для добавления новых и модификации существующих данных, а также для их поиска применяют общий управляемый способ. Данные структурируются таким образом, чтобы была обеспечена возможность дальнейшего наращивания приложений.

Самая жизнеспособная схема организации базы данных была предложена американским комитетом по стандартизации ANSI (American National Standards Institute). Схема предлагает трехуровневую систему организации базы данных (рис.1.1).

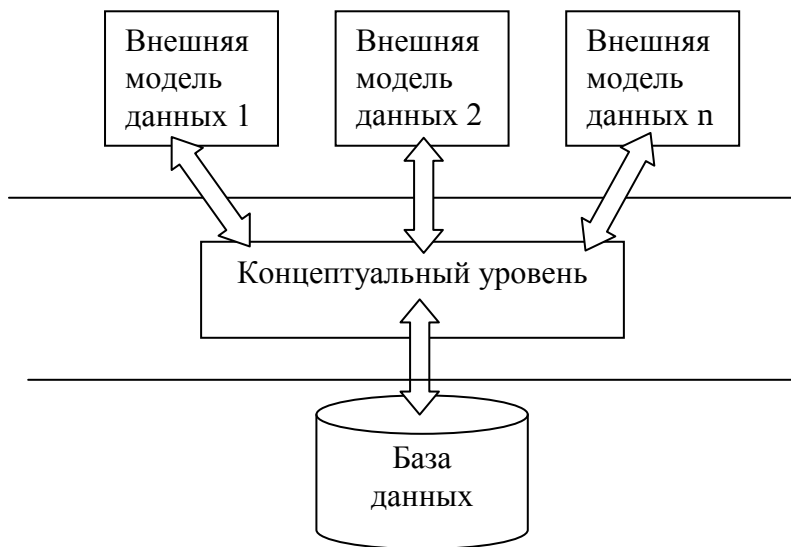


Рис.1.1. Трехуровневая система организации базы данных

Таким образом, существуют три уровня абстракции в представлении базы данных: представление (внешняя модель данных), концептуальная база данных (КБД) и физическая база данных (ФБД). Реально существует только физическая база данных.

Представления и КБД необходимы для реализации процесса проектирования базы данных, для определения структуры данных, используемых при решении локальных задач и для функционирования банка данных в целом.

Представление (внешняя модель данных) – абстрактная информационная модель некоторой части предметной области. Это часть концептуальной базы данных, описание с помощью набора абстракций структуры данных, необходимых для удовлетворения информационных потребностей отдельных групп пользователей и для решения локальных или взаимосвязанных задач.

Этот уровень определяет точку зрения на базу данных отдельных приложений. Каждое приложение видит и обрабатывает только те данные, которые необходимы именно этому приложению. Например, система распределения работ использует сведения о квалификации сотрудника, но ее не интересуют сведения об окладе, домашнем адресе и телефоне сотрудника, и наоборот, именно эти сведения используются в подсистеме отдела кадров.

Концептуальная база данных – информационная модель предметной области банка данных. Это описание с помощью набора абстракций структуры всех данных, необходимых для решения всего комплекса задач в рамках предметной области конкретного банка данных.

Концептуальная база данных определяет концептуальное представление данных в системе. Это представление должно быть по возможности стабильным. В то время как представления отдельных групп пользователей о данных и физическое хранение и организация данных меняются, концептуальная база данных остается неизменной или увеличивается с целью вклю-

чения дополнительных типов данных. Концептуальная база данных определяет общую организацию данных, на основании которой могут быть получены различные внешние организации данных (представления) для отдельных групп пользователей. Общее и внешние логические представления данных являются полностью не зависимыми от физической организации данных.

На концептуальном уровне база данных представлена в наиболее общем виде, который объединяет данные, используемые всеми приложениями, работающими с данной базой данных. Фактически концептуальный уровень отражает обобщенную модель предметной области (объектов реального мира), для которой создавалась база данных. Как любая модель, концептуальная модель отражает только существенные, с точки зрения обработки, особенности объектов реального мира.

Физическая база данных – физическое представление данных и расположение их на запоминающихся устройствах. Это собственно данные, расположенные в файлах или в страничных структурах на внешних носителях информации. Она зависит от используемых средств физического поиска записей. Физическая база данных возникает при решении задач рациональной организации базы данных на запоминающих устройствах.

Организация данных в виде базы обеспечивает логическую и физическую независимость данных. Общая логическая структура данных, являясь величиной относительно постоянной, тем не менее, может меняться.

Логическая независимость данных означает, что изменение общей логической структуры данных (т.е. КБД) не влияет на работу приложений и пользователей. Это обеспечивается тем, что приложения и пользователи работают с базой данных на уровне представлений. Появление новых представлений не затрагивает старых.

Возможны изменения, которые заключаются в удалении данных из базы. В этом случае меняются некоторые представления и требуется коррекция работающих приложений и поль-

зовательских запросов и отчетов. Итак, логическая независимость данных существует между первым и вторым уровнями и означает, что возможно изменение одного приложения без корректировки других приложений, работающих с этой же базой данных.

Физическая независимость данных означает, что физическое расположение и организация данных могут изменяться, не вызывая при этом изменений ни общей логической структуры данных, ни представлений, а значит, и приложений и работы пользователей. Пользователи баз данных не занимаются проблемами представления данных на физическом уровне: размещение данных в памяти, методы доступа к ним и т.д. Такая независимость достигается поддерживаемым СУБД многоуровневым представлением данных в БД на логическом (пользовательском) и физическом уровнях.

Благодаря СУБД и наличию логического уровня представления данных обеспечивается отделение концептуальной (понятийной) модели БД от ее физического представления в памяти компьютера. Итак, физическая независимость данных существует между вторым и третьим уровнями и предполагает возможность переноса хранимой информации с одних носителей на другие при сохранении работоспособности всех приложений, работающих с данной базой данных.

Описание представления называют *подсхемой*, описание КБД – *концептуальной схемой*, описание ФБД – физической схемой. **Схема** – это перечень объектов, их свойств и связей между свойствами и объектами, информация о которых накапливается и обрабатывается в базе данных соответствующего уровня.

Ядром любой базы данных является модель данных. **Модель данных** представляет собой множество структур данных, ограничений целостности и операций манипулирования данными. С помощью модели данных могут быть представлены объекты предметной области и взаимосвязи между ними. **Модель**

данных – совокупность структур данных и операций их обработки.

1.4. Системы управления базами данных

Система управления базой данных – это специальный пакет программ, посредством которого реализуется централизованное управление базой данных и обеспечивается доступ к данным.

Классификация СУБД. В общем случае под СУБД можно понимать любой программный продукт, поддерживающий процессы создания, ведения и использования базы данных.

К СУБД относятся следующие основные виды программ:

- полнофункциональные СУБД;
- серверы баз данных;
- клиенты баз данных;
- средства разработки программ работы с базой данных.

Полнофункциональные СУБД представляют собой традиционные СУБД. Большинство современных СУБД являются полнофункциональными. К ним относятся такие пакеты, как dBaseIV, Microsoft Access, Microsoft FoxPro, Paradox R:BASE.

Обычно полнофункциональные СУБД имеют развитый интерфейс, позволяющий с помощью команд меню выполнять основные действия с базой данных: создавать и модифицировать структуры таблиц, вводить данные, формировать запросы, разрабатывать и выводить на печать отчеты. Для создания запросов и отчетов не обязательно программирование, удобно пользоваться мастерами, а для создания запросов использовать язык QBE (Query By Example – создание запросов по образцу).

Многие полнофункциональные СУБД включают средства программирования для профессиональных разработчиков.

Некоторые системы имеют средства проектирования схем баз данных или CASE-подсистемы, а также средства доступа к другим базам данных и к данным SQL-серверов.

Серверы баз данных предназначены для организации центров обработки данных в сетях ЭВМ. Число таких СУБД постоянно растет.

Эффективность функционирования информационной системы во многом зависит от ее архитектуры. В настоящее время перспективной является архитектура **клиент-сервер**. Она предполагает наличие компьютерной сети и распределенной базы данных, включающей корпоративную базу данных (КБД) и персональные базы данных (ПБД). КБД размещается на компьютере-сервере, ПБД на компьютерах сотрудников подразделений, являющихся клиентами корпоративной базы данных.

Сервером определенного ресурса в компьютерной сети называется компьютер (программа), управляющий этим ресурсом, **клиентом** – компьютер (программа), использующий этот ресурс. В качестве ресурса компьютерной сети могут выступать базы данных, файловые системы, службы печати, почтовые службы. Тип сервера определяется видом ресурса, которым он управляет. Например, если управляемым ресурсом является база данных, то соответствующий сервер называется сервером базы данных. Преимущества организации информационной системы по архитектуре клиент-сервер является удачное сочетание централизованного хранения, обслуживания и коллективного доступа к общей корпоративной информации с индивидуальной работой пользователей над персональной информацией.

Примерами серверов баз данных являются следующие программы: Net Ware SQL (Novell), MS SQL Server (Microsoft), InterBase (Borland), SQLBase Server (Gupta), Intelligent Database (Ingress).

Клиентскими программами для серверов баз данных могут быть различные программы: полнофункциональные СУБД, электронные таблицы, текстовые процессоры, программы электронной почты и т.д. При этом элементы пары «клиент-сервер» могут принадлежать одному или разным производителям программного обеспечения.

Например, для сервера баз данных SQL Server (Microsoft) в роли клиентских программ могут выступать многие СУБД, такие как: MS Access, Visual FoxPro, dBaseIV, Blyth Software, Paradox, Focus и др.

Средства разработки программ работы с базами данных могут использоваться для создания разновидностей следующих программ:

- клиентских программ;
- серверов баз данных и их отдельных компонентов;
- пользовательских приложений.

Программы первого и второго видов довольно малочисленны, так как предназначены, главным образом, для системных программистов. Пакетов третьего вида гораздо больше, но меньше, чем полнофункциональных СУБД.

К средствам разработки пользовательских приложений относятся системы программирования, разнообразные библиотеки программ для различных языков программирования, а также пакеты автоматизации разработок (в том числе систем типа клиент-сервер). В числе наиболее распространенных можно назвать следующие инструментальные системы: Delphi и Power Builder (Borland), Visual Basic (Microsoft), SILVERRUN (Computer Advisers Inc.), S-Designor (SDP и Powersoft) и Erwin (Logic Works).

По характеру использования СУБД делятся на персональные и многопользовательские.

Персональные СУБД обычно обеспечивают возможность создания персональных баз данных и недорогих приложений, работающих с ними. Персональные СУБД или разработанные с их помощью приложения могут выступать в роли клиентской части многопользовательской СУБД. К персональным СУБД, например, относятся Visual FoxPro, Paradox, Clipper, dBase, Access и др.

Многопользовательские СУБД включают в себя сервер баз данных и клиентскую часть и, как правило, могут работать в неоднородной вычислительной среде (с разными типами ЭВМ и

операционными системами). К многопользовательским СУБД относятся, например, СУБД Oracle и Informix.

По используемой модели данных СУБД (как и базы данных) разделяются на *иерархические, сетевые, реляционные, объектно-ориентированные и другие типы*. Некоторые СУБД могут одновременно поддерживать несколько моделей данных.

С точки зрения пользователя, СУБД реализует **функции** хранения, изменения (добавления, редактирования и удаления) и обработки информации, а также разработки и получения различных выходных документов.

Для работы с хранящейся в базе данных информацией СУБД предоставляет программам и пользователям следующие **два типа языков**:

- **язык описания данных** – высокоуровневый непроецедурный язык декларативного типа, предназначенный для описания логической структуры данных;

- **язык манипулирования данными** – совокупность конструкций, обеспечивающих выполнение основных по работе с данными: ввод, удаление, модификацию и выборку данных по запросам.

Названные языки в различных СУБД могут иметь отличия. Наибольшее распространение получили два стандартных языка: QBE (Query By Example) – язык запросов по образцу и SQL (Structured Query Language) – структурированный язык запросов. QBE в основном обладает свойствами *языка манипулирования данными*, SQL сочетает в себе свойства языков обоих типов – *описания и манипулирования данными*.

СУБД также выполняет **функции**, которые называют **низкоуровневыми**:

- управление данными во внешней памяти;
- управление буферами оперативной памяти;
- управление транзакциями;
- ведение журнала изменений в базе данных;
- обеспечение целостности и безопасности базы данных.

Методы и алгоритмы **управления данными** являются «внутренним делом» СУБД и прямого отношения к пользователю не имеют. Качество реализации этой функции наиболее сильно влияет на эффективность работы специфических информационных систем, например, работающих с огромными базами данных, со сложными запросами, осуществляющих большие объемы обработки данных.

Буферизация данных и как следствие реализация **функции управления буферами оперативной памяти** обусловлена тем, что объем оперативной памяти меньше объема внешней памяти.

Буферы представляют собой области оперативной памяти, предназначенные для ускорения обмена данными между внешней и оперативной памятью. В буферах временно хранятся фрагменты базы данных, данные из которых предполагается использовать при обращении к СУБД или планируется записать в базу данных после обработки.

Механизм транзакций используется в СУБД для поддержания целостности данных в базе. **Транзакцией** называется некоторая неделимая последовательность операций над данными базы данных, которая отслеживается СУБД от начала до завершения. Если по каким-либо причинам (сбои и отказы оборудования, ошибки в программном обеспечении, включая приложение) транзакция остается незавершенной, то она отменяется.

Транзакции присущи три основных свойства:

- атомарность (выполняются все входящие в транзакцию операции или ни одна);
- сериализуемость (отсутствует взаимное влияние выполняемых в одно и то же время транзакций);
- долговечность (даже крах системы не приводит к утрате результатов зафиксированной транзакции).

Примером транзакций является операция перевода денег с одного счета на другой в банковской системе. Здесь необходим, по крайней мере, двухшаговый процесс. Сначала снимают деньги с одного счета, затем добавляют их к другому счету. Если хотя бы одно из действий не выполнится успешно, результат

операции окажется неверным и будет нарушен баланс между счетами.

Контроль транзакций важен в многопользовательских СУБД, где транзакции могут быть запущены параллельно. В последнем случае говорят о сериализуемости транзакций. Под **сериализацией** параллельно выполняемых транзакций понимается составление такого плана их выполнения (сериального плана), при котором суммарный эффект реализации транзакций эквивалентен эффекту их последовательного выполнения.

При параллельном выполнении нескольких транзакций возможно возникновение конфликтов (блокировок), разрешение которых является функцией СУБД. При обнаружении таких случаев обычно производится «откат» путем отмены изменений, произведенных одной или несколькими транзакциями.

Ведение журнала изменений в базе данных (журнализация изменений) выполняется СУБД для обеспечения надежности хранения данных в базе при наличии аппаратных сбоев и отказов, а также ошибок в программном обеспечении.

Журнал СУБД – это особая база данных или часть основной базы данных, непосредственно недоступная пользователю и используемая для записи информации обо всех изменениях базы данных. В различных СУБД в журнал могут заноситься записи, соответствующие изменениям в СУБД на разных уровнях: от минимальной внутренней операции модификации страницы внешней памяти до логической операции модификации базы данных (например, вставки записи, удаления столбца, изменения значения в поле) и даже транзакции.

Для эффективной реализации функции ведения журнала изменений в базе данных необходимо обеспечить повышенную надежность хранения и поддержания в рабочем состоянии самого журнала. Иногда для этого в системе хранят несколько копий журнала.

Обеспечение целостности базы данных составляет необходимое условие успешного функционирования базы данных, особенно для случая использования базы данных в сетях. **Цело-**

стность базы данных есть свойство базы данных, означающее, что в ней содержится полная, непротиворечивая и адекватно отражающая предметную область информация. Поддержание целостности базы данных включает проверку целостности и ее восстановление в случае обнаружения противоречий в базе данных. Целостное состояние базы данных описывается с помощью ограничений целостности в виде условий, которым должны удовлетворять хранимые в базе данные. Примером таких условий может служить ограничение диапазонов возможных значений атрибутов объектов, сведения о которых хранятся в базе данных, или отсутствие повторяющихся записей в таблицах реляционных баз данных.

Обеспечение безопасности достигается в СУБД шифрованием прикладных программ, данных, защиты паролем, поддержкой уровней доступа к базе данных и к отдельным ее элементам (таблицам, формам, отчетам и т.д.).

1.5. Оперативные и аналитические системы

Базы данных можно использовать для оперативной (текущей) и аналитической обработки информации.

Первоначально базы данных использовались в оперативных системах обработки информации. Данные системы за рубежом получили название On-Line Transaction Processing (OLTP). Общая схема организации OLTP представлена на рис. 1.2.

К середине 90-х годов XX века в базах данных класса OLTP скопилось столько хронологической информации, что объем БД резко возрос, а быстродействие начало падать. Стало ясно, что ретроспективную информацию следует периодически передавать в отдельную БД. К тому же ретроспективные данные обладают новым качеством: они позволяют вырабатывать стратегические решения. Возникла необходимость в системах поддержки принятия стратегических решений (СППР). Такие

системы получили за рубежом название On-Line Analytical Processing (OLAP).

OLAP использует многомерное представление агрегированных данных для быстрого доступа к важной информации и дальнейшего ее анализа.

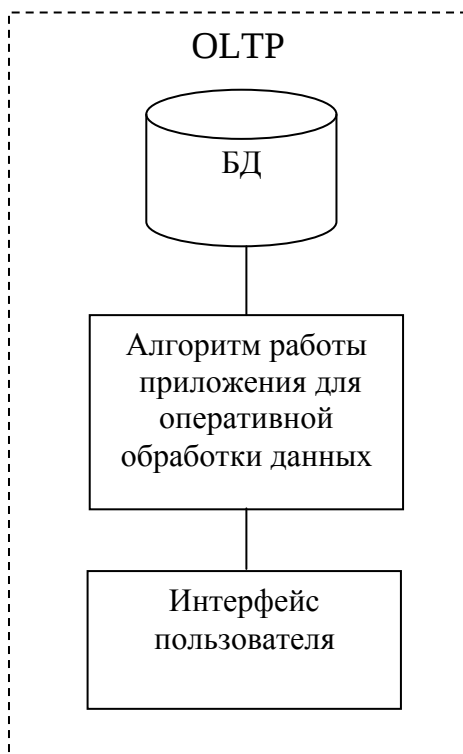


Рис. 1.2. Схема организации OLTP

Системы OLAP обеспечивают аналитикам и руководителям быстрый интерактивный доступ к внутренней структуре данных и возможность преобразования исходных данных с тем, чтобы они позволяли отразить структуру системы нужным для пользователя способом. Кроме того, OLAP-системы позволяют

просматривать данные и выявлять имеющиеся в них закономерности либо визуально, либо простейшими методами (такими как линейная регрессия), а включение в их арсенал нейросетевых методов обеспечивает существенное расширение аналитических возможностей.

В основе концепции оперативной аналитической обработки (OLAP) лежит многомерное представление данных. Термин OLAP ввел Э.Ф. Кодд (E.F. Codd) в 1993 году. В своей статье он рассмотрел недостатки реляционной модели данных, в первую очередь невозможность «объединять, просматривать и анализировать данные с точки зрения множественности измерений, то есть самым понятным для корпоративных аналитиков способом». Кодд определил общие требования к системам OLAP, расширяющим функциональность реляционных СУБД и включающим многомерный анализ как одну из своих характеристик.

Следует заметить, что Кодд обозначает термином OLAP многомерный способ представления данных исключительно на концептуальном уровне. Используемые им термины – «Многомерное концептуальное представление» (“Multidimensional conceptual view”), «Множественные измерения данных» (“Multiple data dimensions”), «Сервер OLAP» (“OLAP server”) – не определяют физического механизма хранения данных. По Кодду, многомерное концептуальное представление (multi-dimensional conceptual view) является наиболее естественным взглядом управляющего персонала на объект управления. Оно представляет собой множественную перспективу, состоящую из нескольких независимых измерений, вдоль которых могут быть проанализированы определенные совокупности данных. Одновременный анализ по нескольким измерениям данных определяется как многомерный анализ. Каждое измерение включает направления консолидации данных, состоящие из серии последовательных уровней обобщения, где каждый вышестоящий уровень соответствует большей степени агрегации данных по соответствующему измерению. Так, измерение Исполнитель может определяться направлением консолидации, состоящим из уровней

обобщения «предприятие – подразделение – отдел - служащий». Измерение Времени может включать два направления консолидации – «год – квартал – месяц - день» и «неделя - день», поскольку счет времени по месяцам и по неделям несовместим. В этом случае становится возможным произвольный выбор желаемого уровня детализации информации по каждому из измерений. Операция спуска (drilling down) соответствует движению от высших ступеней консолидации к низшим; напротив, операция подъема (rolling up) означает движение от низших уровней к высшим.

Общая схема организации OLAP представлена на рис. 1.3.

Хранилище данных – предметно-ориентированный, интегрированный, привязанный ко времени и неизменный набор данных, предназначенный для поддержки принятия решений.

Интегрированность означает, что источниками данных могут быть несколько БД, данные в которых различаются форматами и степенью заполнения данных. Эти данные должны быть приведены к «стандарту», используемому в хранилище данных.

Привязка ко времени означает, что исходные данные характеризуют какой-то интервал времени, при этом время присутствует в БД явно. В силу этого вновь поступающие данные не изменяют прежние данные в ХД, а дополняют их.

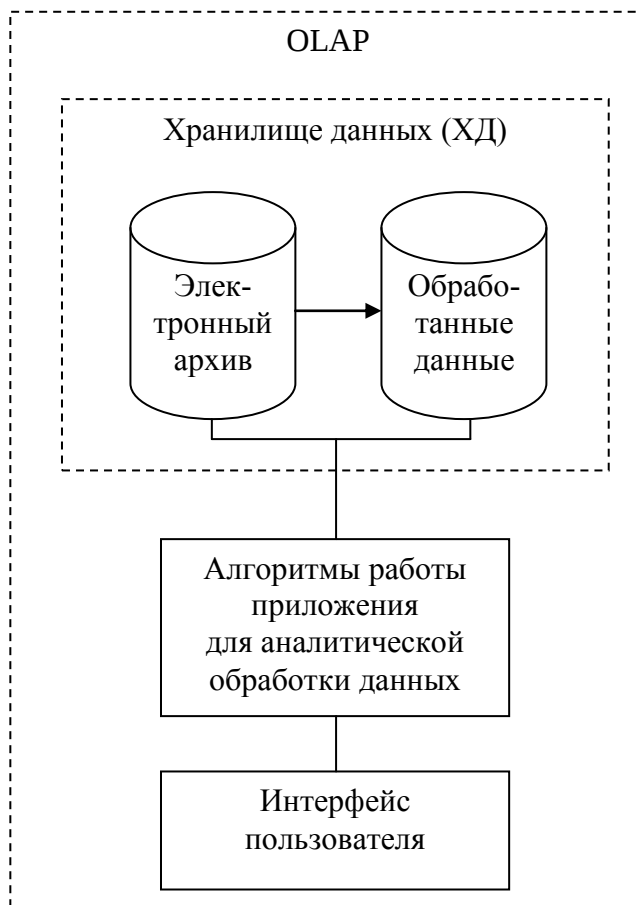


Рис. 1.3. Схема организации OLAP

Сравнительные характеристики OLTP и OLAP приведены в табл. 1.1.

Свойство	OLTP	OLAP
Назначение данных	Оперативный поиск, несложная обработка	Аналитическая обработка: прогнозирование, моделирование, анализ и выявление связей, выявление статистических закономерностей
Уровень агрегации данных	Детальные данные	Агрегированные данные
Период хранения данных	Несколько лет	Несколько десятков лет
Изменчивость данных	Изменяются, добавляются, удаляются	Добавляются
Упорядочение данных	По любому полю	По хронологии
Объем обрабатываемой информации	Средний	Очень большой
Скорость обработки	Средняя	Очень высокая
Критерий эффективности работы	Количество транзакций в единицу времени	Скорость выполнения сложных запросов
Загрузка	Часто, небольшими и средними порциями	Редко и очень большими порциями

В составе OLAP можно выделить следующие функциональные элементы (рис. 1.4):

- хранилище данных (ХД);
- менеджер загрузки;

- менеджер хранилища данных;
- менеджер запросов.

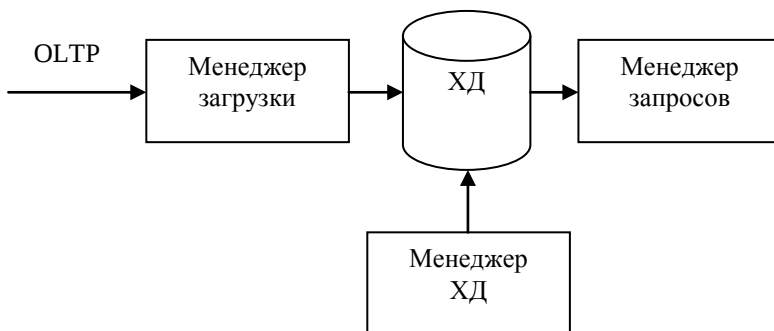


Рис. 1.4. Функциональные элементы OLAP

Хранилище данных можно считать базой данных, в которой выделяют электронный архив, хранящий детальные ретроспективные данные, и агрегированные (обработанные данные).

ХД реализуется с помощью многомерной модели, которая имеет несколько разновидностей: собственно многомерная (Multidimensional OLAP – MOLAP), реляционная (Relational OLAP – ROLAP), гибридная (Hibrid OLAP – HOLAP).

Менеджер загрузки осуществляет преобразование данных, поступающих из операционных БД, и прежде всего форматирование по «стандарту» OLAP.

Менеджер хранилища данных выполняет следующие операции:

- анализ непротиворечивости исходных данных;
- создание необходимых индексов и видов;
- денормализацию;
- резервное копирование.

Менеджер запросов управляет пользовательскими запросами.

1.6. Требования, предъявляемые к базам данных

К транзакционным (оперативным) базам данных и соответствующим СУБД предъявляются следующие требования.

1. Простота обновления данных. Под операцией обновления понимают добавление, удаление и изменение данных.

2. Высокое быстродействие (малое время отклика на запрос). Время отклика – промежуток времени от момента запроса к БД и фактическим получением данных. Похожим является термин время доступа – промежуток времени между выдачей команды записи (считывания) и фактическим получением данных. Под доступом понимается операция поиска, чтения данных или записи их.

3. Независимость данных. Независимость данных – это возможность изменения логической и физической структуры БД без изменения представлений пользователей. Независимость данных предполагает инвариантность к характеру хранения данных, программному обеспечению и техническим средствам. Она обеспечивает минимальные изменения структуры БД при изменениях стратегии доступа к данным и структуры самих исходных данных.

4. Совместное использование данных многими пользователями.

5. Безопасность данных – защита данных от преднамеренного или непреднамеренного нарушения секретности, искажения или разрушения.

Безопасность данных включает их целостность и защиту. Целостность данных – устойчивость хранимых данных к разрушению и уничтожению, связанных с неисправностями технических средств, системными ошибками и ошибочными действиями пользователей.

Она предполагает:

- отсутствие неточно введенных данных или двух одинаковых записей об одном и том же факте;
- защиту от ошибок при обновлении БД;

- невозможность удаления данных из связанных таблиц;
- неискажение данных при работе в многопользовательском режиме и в распределенных базах данных;
- сохранность данных при сбоях техники (восстановление данных).

Целостность обеспечивается триггерами целостности – специальными приложениями – программами, работающими при определенных условиях. Для некоторых СУБД (например, Access, Paradox) триггеры являются встроенными.

Защита данных от несанкционированного доступа предполагает ограничение доступа к конфиденциальным данным. Защиту данных могут обеспечивать следующие действия:

- введение системы паролей;
- разграничение доступа к данным через специальные настройки, осуществляемые администратором базы данных;
- формирование видов – специальных таблиц, производных от исходных и предназначенных конкретным пользователям.

6. Стандартизация построения и эксплуатации БД (СУБД).

Стандартизация обеспечивает преемственность поколений СУБД, упрощает взаимодействие БД одного поколения СУБД с одинаковыми и различными моделями данных. Стандартизация (ANSI/SPARC) осуществлена в значительной степени в части интерфейса пользователя и языка SQL. Это позволило успешно решить задачу взаимодействия различных реляционных СУБД как с помощью языка SQL, так и с применением приложения Open DataBase Connection (ODBC). При этом может быть осуществлен как локальный, так и удаленный доступ к данным.

7. Адекватность отображения данных соответствующей предметной области.

8. Дружественный интерфейс пользователя.

Данные из операционной БД периодически передаются в электронный архив хранилища данных для последующего анализа и обработки. Поскольку данные в хранилище практически

не изменяются, а лишь добавляются, требование простоты обновления становится неактуальным. На первое место – в силу значительного объема данных в хранилище – выходит требование высокого быстродействия.

К хранилищам данных предъявляются следующие дополнительные требования:

- высокая производительность загрузки данных из операционных БД;
- возможность фильтрования, переформатирования, проверки целостности исходных данных, индексирования данных, обновления метаданных;
- повышенные требования к непротиворечивости исходных данных, поскольку они могут быть получены из разных источников;
- высокая скорость выполнения запросов;
- обеспечение высокой размерности;
- одновременность доступа к ХД;
- наличие средств администрирования;
- поддержка анализа данных соответствующими методами (инструментами).

Э.Ф. Кодд на основе своего опыта предъявил следующие требования к системе OLAP.

1. Многомерное концептуальное представление данных.
2. Прозрачность технологии и источников данных.
3. Доступность к источникам данных при использовании различных моделей данных.
4. Неизменная производительность подготовки отчетов при росте Объема, количества измерений, процедур обобщения данных.
5. Использование гибкой, адаптивной, масштабируемой архитектуры клиент-сервер.
6. Универсальность измерений (формулы и средства создания отчетов не должны быть привязаны к конкретным видам размерностей).

7. Динамическое управление разреженностью матриц (пустые значения NULL должны храниться эффективным образом).
8. Многопользовательская поддержка.
9. Неограниченные операционные связи между размерностями.
10. Поддержка интуитивно понятных манипуляций с данными.
11. Гибкость средств формирования отчетов.
12. Неограниченное число измерений и уровней обобщения.

2. МОДЕЛИ ДАННЫХ

Хранимые в базе данные имеют определенную логическую структуру, описываются некоторой моделью представления данных (моделью данных), поддерживаемой СУБД.

Модель данных (модель представления данных) – это множество элементов (объектов, типов данных) и связей между ними, ограничений (например, целостности, авторизации, синхронизации многопользовательского доступа) и операций над объектами, типами данных и связями.

Множество допустимых типов данных и их отношений образуют структуру данных. В моделях данных, таким образом, выделяют три компоненты: структура данных; ограничения, определяющие допустимые состояния БД; множество операций, применяемых для поиска и обновления данных.

К числу классических относятся следующие модели данных: иерархическая; сетевая; реляционная.

Кроме того, в последние годы появились и стали более активно внедряться на практике следующие модели данных: многомерная, объектно-ориентированная.

Разрабатываются также всевозможные системы, основанные на других моделях данных, расширяющих известные модели. В их числе можно назвать объектно-реляционные, дедук-

тивно-объектно-ориентированные, семантические, концептуальные и ориентированные модели. Некоторые из этих моделей служат для интеграции баз данных, баз знаний и языков программирования. В некоторых СУБД поддерживается одновременно несколько моделей данных.

2.1. Иерархическая модель данных

В иерархической модели данные можно описать с помощью упорядоченного графа (или дерева). Упрощенно представление связей между данными в иерархической модели показано на рис. 2.1.

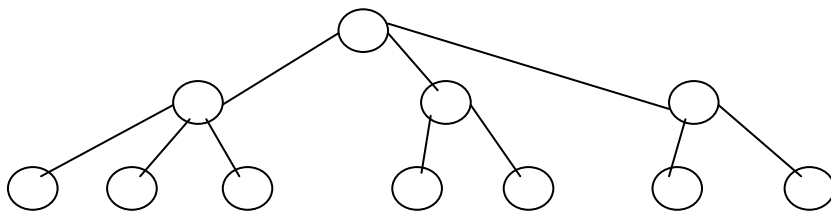


Рис. 2.1. Представление связей в иерархической модели

Для описания структуры (схемы) иерархической БД на некотором языке программирования используется тип данных «дерево».

Тип «дерево» схож с типом данных «запись» языка Паскаль. Допускается вложенность типов, каждый из которых находится на некотором уровне.

Тип «дерево» является составным. Он включает в себя подтипы («поддережья»), каждый из которых, в свою очередь, является типом «дерево». Каждый из типов «дерево» состоит из одного «корневого» типа и упорядоченного набора (возможно пустого) подчиненных типов. Каждый из элементарных типов, включенных в тип «дерево», является простым или составным типом «запись». Простая «запись» состоит из одного типа, например, числового. Составная «запись» объединяет некоторую

совокупность типов, например, целое, строку символов и указатель (ссылку). Пример типа «дерево» как совокупности типов показан на рис. 2.2.

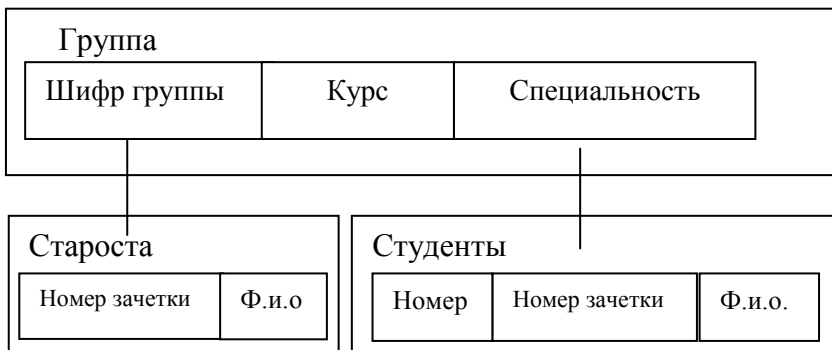


Рис. 2.2. Пример типа «дерево»

Корневым называется тип, который имеет подчиненные типы и сам не является подтипом. **Подчиненный тип** (подтип) является потомком по отношению к типу, который выступает для него в роли предка (родителя). Потомки одного и того же типа являются близнецами по отношению друг к другу.

В целом тип «дерево» представляет собой иерархически организованный набор типов «запись».

Иерархическая база данных представляет собой упорядоченную совокупность экземпляров данных типа «дерево» (деревьев), содержащих экземпляры типа «запись» (записи). Часто отношения родства между типами переносят на отношения между самими записями. Поля записей хранят собственно числовые или символьные значения, составляющие основное содержание БД. Обход всех элементов иерархической БД обычно производится сверху вниз и слева направо.

В иерархической СУБД может использоваться терминология, отличающаяся от приведенной. Например, запись могут называть сегментом, а под записью БД понимать всю совокуп-

ность записей, относящихся к одному экземпляру типа "«дерево»". Данные в базе с приведенной схемой (рис.2.2.) могут выглядеть, например, как показано на рис.2.3.

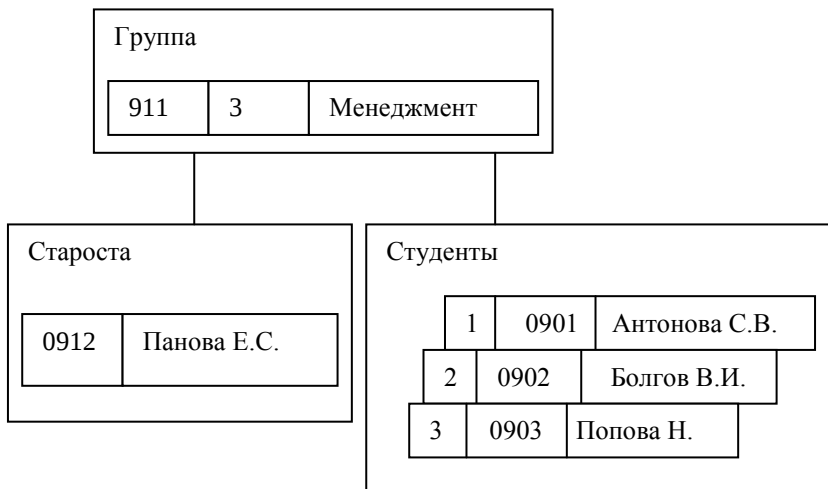


Рис. 2.3. Данные в иерархической базе

Для организации физического размещения иерархических данных в памяти компьютера могут использоваться следующие группы методов:

- представление линейным списком с последовательным распределением памяти (адресная арифметика, левосписковые структуры);
- представление связными линейными списками (методы, использующие указатели и справочники).

К основным операциям манипулирования иерархически организованными данными относятся следующие:

- поиск указанного экземпляра БД (например, поиск дерева со значением 912 в поле Шифр_группы);
- переход от одного дерева к другому;

- переход от одной записи к другой внутри дерева (например, к следующей записи типа Студенты);
- вставка новой записи в указанную позицию;
- удаление текущей записи и т.д.

В соответствии с определением типа «дерево», можно заключить, что между предками и потомками автоматически поддерживается контроль целостности связей. Основное правило контроля целостности формулируется следующим образом: потомок не может существовать без родителя, а у некоторых родителей может не быть потомков. Механизмы поддержания целостности связей между записями различных деревьев отсутствуют.

К достоинствам иерархической модели данных относятся эффективное использование памяти компьютера и неплохие показатели времени выполнения основных операций над данными. Иерархическая модель данных удобна для работы с иерархически упорядоченной информацией.

Недостатком иерархической модели является ее громоздкость для обработки информации с достаточно сложными логическими связями, а также сложность понимания для обычного пользователя.

На иерархической модели данных основано сравнительно ограниченное количество СУБД, в числе которых можно назвать зарубежные системы IMS, PS/Focus, Team-Up, Data Edge, а также отечественные системы Ока, ИНЭС, МИРИС.

2.2. Сетевая модель

Сетевая модель данных позволяет отображать разнообразные взаимосвязи элементов данных в виде произвольного графа, обобщая тем самым иерархическую модель данных (рис. 2.4). Наиболее полно концепция сетевых БД впервые была изложена в Предложениях группы КОДАСИЛ (KODASYL).

Для описания схемы сетевой БД используются две группы типов: «запись» и «связь». Тип «связь» определяется для двух

типов «запись»: предка и потомка. Переменные типа «связь» являются экземплярами связей.

Сетевая БД состоит из набора записей и набора соответствующих связей. На формирование связи особых ограничений не накладывается. Если в иерархических структурах запись-потомок могла иметь только одну запись-предок, то в сетевой модели данных запись-потомок может иметь произвольное число записей-предков (сводных родителей).

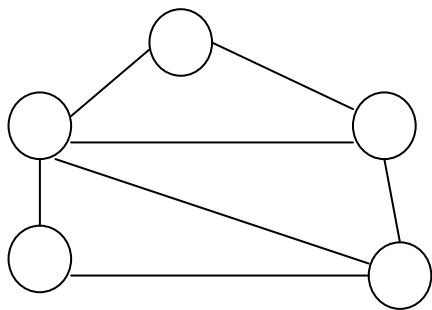


Рис. 2.4. Представление связей в сетевой модели

Пример схемы простейшей сетевой БД показан на рис. 2.5. Типы связей здесь обозначены надписями на соединяющих типы записей линиях.

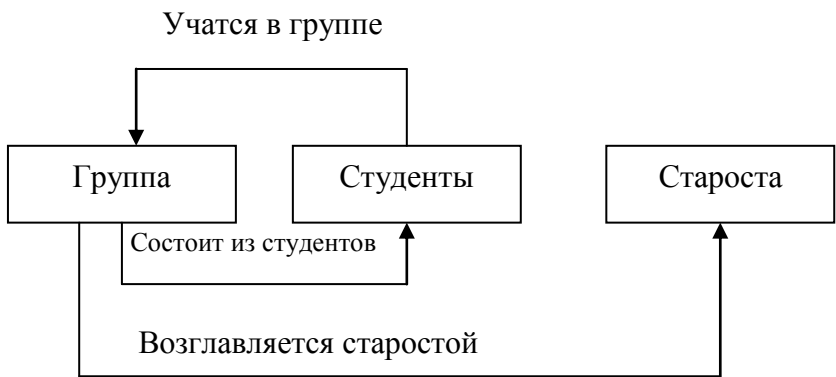


Рис. 2.5. Пример схемы сетевой БД

В различных СУБД сетевого типа для обозначения одинаковых по сути понятий могут использоваться различные термины. Например, такие, как элементы и агрегаты данных, записи, наборы, области и т.д.

Физическое размещение данных в базах сетевого типа может быть организовано практически теми же методами, что и в иерархических базах данных.

К числу важнейших операций манипулирования данными баз сетевого типа можно отнести следующие операции:

- поиск записи в БД,
- переход от предка к первому потомку,
- переход от потомка к предку,
- создание новой записи,
- удаление текущей записи,
- обновление текущей записи,
- включение записи в связь,
- исключение записи из связи,
- изменение связей и т.д.

Достоинством сетевой модели данных является возможность эффективной реализации по показателям затрат времени и оперативности. В сравнении с иерархической моделью сетевая модель предоставляет большие возможности в смысле допустимости образования произвольных связей.

Недостатком сетевой модели данных является высокая сложность и жесткость схемы БД, построенной на ее основе, а также сложность для понимания и выполнения операций обработки данных в БД для обычных пользователей. Кроме того, в сетевой модели данных ослаблен контроль целостности связей вследствие допустимости установления произвольных связей между записями.

Системы на основе сетевой модели не получили широкого распространения на практике. Наиболее известными сетевыми СУБД являются следующие: IDMS, db_VistaIII, СЕТЬ, СЕТОР и КОМПАС.

2.3. Реляционная модель

Реляционная модель данных была предложена сотрудником фирмы IBM Эдгаром Коддом и основывается на понятии отношение (relation).

Отношение представляет собой множество элементов, называемых кортежами. Подробно теоретическая основа реляционной модели рассматривается в следующем разделе. Наглядной формой представления отношения является привычная для человеческого восприятия двумерная таблица.

Таблица имеет строки (записи) и столбцы (колонки). Каждая строка таблицы имеет одинаковую структуру и состоит из полей. Строкам таблицы соответствуют кортежи, а столбцам – атрибуты отношения.

С помощью одной таблицы удобно описывать сведения о группах однородных (имеющих одинаковые свойства) объектов, явлений или процессов реального мира. Каждая строка таблицы содержит сведения о конкретном объекте, явлении или процессе. Строка (запись) имеет одинаковую структуру и описывает с помощью полей свойства объектов. Например, таблица может содержать сведения о группе обучаемых, о каждом из которых известны следующие характеристики: фамилия, имя и отчество, пол, дата рождения, адрес проживания. Поскольку в рамках одной таблицы не удастся описать все данные из предметной области, то создается несколько таблиц, между которыми устанавливаются связи.

Физическое размещение данных в реляционных базах на внешних носителях легко осуществляется с помощью обычных файлов.

Преимущества реляционной модели данных заключаются в простоте, понятности и удобстве физической реализации на ЭВМ. Именно простота и понятность для пользователя явились основной причиной их широкого использования.

Основными *недостатками* реляционной модели являются следующие: отсутствие стандартных средств идентификации

отдельных записей и сложность описания иерархических и сетевых связей.

Примерами реляционных СУБД являются следующие: dBaseIIIPlus dBaseIV (фирма Ashton-Tate), DB2 (IBM), R:BASE (Microrim), FoxPro ранних версий и FoxBase (Fox Software), Paradox и dBASE for Windows (Borland), FoxPro более поздних версий, Visual FoxPro и Access (Microsoft), Clarion (Clarion Software), Ingres (ASK Computer System) и Oracle (Oracle).

Последние версии реляционных СУБД имеют некоторые свойства объектно-ориентированных систем. Такие СУБД часто называют объектно-реляционными. Примером такой системы можно считать Oracle 8.x.

2.4. Постреляционная модель

Классическая реляционная модель предполагает неделимость данных, хранящихся в полях записей таблиц. Это означает, что информация в таблице представляется в первой нормальной форме. Существует ряд случаев, когда это ограничение мешает эффективной реализации приложений.

Постреляционная модель данных представляет собой расширенную реляционную модель, снимающую ограничение неделимости данных, хранящихся в записях таблиц. Постреляционная модель данных допускает многозначные поля – поля, значения которых состоят из подзначений. Набор значений многозначных полей считается самостоятельной таблицей, встроенной в основную таблицу.

На рис. 2.6 для сравнения приведено представление одних и тех же данных с помощью реляционной (а) и постреляционной (б) моделей.

а) Накладные

Номер накладной	Номер покупателя
0373	8723
8374	8232
7364	8723

Накладные_Товары

Номер накладной	Товар	Количество
0373	Сыр	3
0373	Рыба	2
8374	Шоколад	2
8374	Сок	6
8374	Печенье	2
7364	Йогурт	1

б) Накладные

Номер накладной	Номер покуп-я	Товар	Кол-во
0373	8723	Сыр	3
		Рыба	2
8374	8232	Шоколад	2
		Сок	6
		Печенье	2
7364	8723	Йогурт	1

Рис. 2.6. Структуры данных реляционной и постреляционной моделей

Таблица Накладные содержит данные о номерах накладных и номерах покупателей. В таблице Накладные_Товары содержатся данные о каждой из накладных: номер накладной, название товара и количество товара. Таблица Накладные связана с таблицей Накладные_Товары по полю Номер накладной.

Как видно из рисунка, по сравнению с реляционной моделью в постреляционной модели данные хранятся более эффективно, а при обработке не требуется выполнять операцию соединения данных их двух таблиц.

Помимо обеспечения вложенности полей постреляционная модель поддерживает ассоциированные многозначные поля (множественные группы). Совокупность ассоциированных полей называется ассоциацией. При этом в строке первое значение одного столбца ассоциации соответствует первым значениям всех других столбцов ассоциации. Аналогичным образом связаны все вторые значения столбцов и т.д.

На длину полей и количество полей в записях таблицы не накладывается требование постоянства. Это означает, что структура данных и таблиц имеет большую гибкость.

Поскольку постреляционная модель допускает хранение в таблицах ненормализованных данных, возникает проблема обеспечения целостности и непротиворечивости данных. Эта проблема решается включением в СУБД механизмов, подобных хранимым процедурам в клиент-серверных системах.

Для описания функций контроля значений в полях имеется возможность создавать процедуры (коды конверсии и коды корреляции), автоматически вызываемые до и после обращения к данным. Коды корреляции выполняются сразу после чтения данных, перед их обработкой. Коды конверсии, наоборот, выполняются после обработки данных.

Преимуществом постреляционной модели является возможность представления совокупности связанных реляционных таблиц одной постреляционной таблицей. Это обеспечивает высокую наглядность представления информации и повышение эффективности ее обработки.

Недостатком постреляционной модели является сложность решения проблемы обеспечения целостности и непротиворечивости хранимых данных.

Рассмотренная постреляционная модель данных поддерживается СУБД uniVers, Bubba, Dasdb.

2.5. Многомерная модель

Многомерный подход к представлению данных в базе появился практически одновременно с реляционным, но реально работающих многомерных СУБД (МСУБД) до настоящего времени было очень мало. С середины 90-х годов интерес к ним стал приобретать массовый характер.

Толчком послужила в 1993 году программная статья одного из основоположников реляционного подхода Э. Кодда. В ней сформулированы 12 основных требований к системам класса OLAP (OnLine Analytical Processing – оперативная аналитическая обработка), важнейшие из которых связаны с возможностями концептуального представления и обработки многомерных данных. Многомерные системы позволяют оперативно обрабатывать информацию для проведения анализа и принятия решения.

В развитии концепции ИС можно выделить следующие два направления:

- системы оперативной (транзакционной) обработки;
- системы аналитической обработки (системы принятия решений).

Реляционные СУБД предназначались для информационных систем оперативной обработки информации и в этой области были весьма эффективны. В системах аналитической обработки они показали себя несколько неповоротливыми и недостаточно гибкими. Более эффективными здесь оказываются многомерные СУБД.

Многомерные СУБД являются узкоспециализированными СУБД, предназначенными для интерактивной аналитической обработки информации.

Многомерная модель данных (ММД) используется в хранилищах данных. Имеются три разновидности ММД: MOLAP, ROLAP, HOLAP.

MOLAP предполагает организацию данных в виде так называемого многомерного куба. Каждая из координат (ребер)

куба называется измерением. Значение результата в многомерной модели помещается на пересечении (в ячейке) соответствующих значений измерения.

В ROLAP используются реляционные составляющие. В любой схеме выделяются одна фактологическая таблица, в которой хранятся данные, и несколько справочных таблиц, каждая из которых характеризует одну из размерностей куба.

Сравнение MOLAP и ROLAP дает следующие результаты:

- MOLAP обладает высоким быстродействием, но возникают проблемы с хранением больших объемов данных;
- ROLAP не имеет ограничений на объем данных, однако обладает гораздо меньшим быстродействием.

Отсюда возникает идея построения HOLAP, совмещающего достоинства MOLAP и ROLAP.

Все данные ХД одновременно никогда не требуются. Каждый раз используется лишь их часть. В связи с этим целесообразно данные разделить на предметные подобласти, которые называют киосками (магазинами, витринами) данных. Киоск данных – специализированное тематическое хранилище, обслуживающее одно из направлений деятельности фирмы. В этом случае централизованное хранилище может быть реализовано с использованием реляционной БД, а основные данные хранятся в многочисленных киосках.

Раскроем основные понятия, используемые в многомерных СУБД: агрегируемость, историчность, прогнозируемость данных.

Агрегируемость данных означает рассмотрение информации на различных уровнях ее обобщения. В информационных системах степень детальности представления информации для пользователя зависит от его уровня: аналитик, пользователь-оператор, управляющий, руководитель.

Историчность данных предполагает обеспечение высокого уровня статичности (неизменности) собственно данных и их взаимосвязей, а также обязательность привязки данных ко времени. Статичность данных позволяет использовать при их об-

работке специализированные методы загрузки, хранения, индексации и выборки.

Временная привязка данных необходима для частого выполнения запросов, имеющих значения времени и даты в составе выборки.

Прогнозируемость данных подразумевает задание функций прогнозирования и применение их к различным временным интервалам.

Многомерность модели данных означает не многомерность визуализации цифровых данных, а многомерное логическое представление структуры информации при описании и в операциях манипулирования данными.

По сравнению с реляционной моделью многомерная организация данных обладает более высокой наглядностью и информативностью. Для иллюстрации на рис. 2.7 приведены реляционное (а) и многомерное (б) представления одних и тех же данных об объемах продаж автомобилей.

а)

Модель	Месяц	Объемы
«Жигули»	Июнь	12
«Жигули»	Июль	24
«Жигули»	Август	5
«Москвич»	Июнь	2
«Москвич»	Июль	18
«Волга»	Июль	19

б)

Модель	Июнь	Июль	Август
«Жигули»	12	24	5
«Москвич»	2	18	Нет
«Волга»	Нет	19	Нет

Рис. 2.7. Реляционное и многомерное представления данных

Если речь идет о многомерной модели с мерностью больше двух, то не обязательно визуальное представление

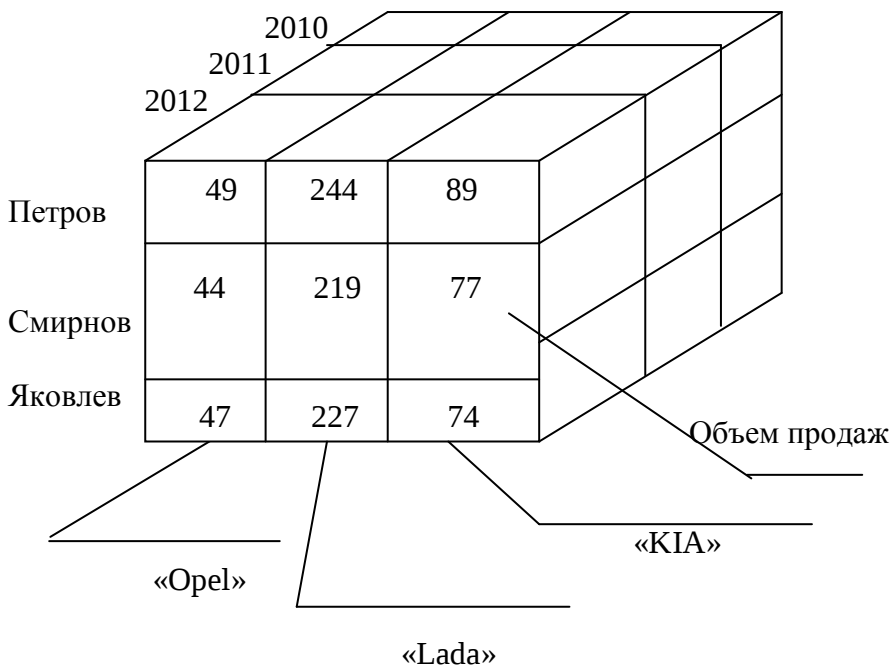
ется в виде многомерных объектов (трех-, четырех- и более мерных гиперкубов). Пользователю и в этих случаях более удобно иметь дело с двумерными таблицами или графиками. Данные при этом представляют собой «вырезки» (точнее «срезы») из многомерного хранилища данных, выполненные с разной степенью детализации.

Рассмотрим основные понятия многомерных моделей данных, к числу которых относятся измерение и ячейка.

Измерение (Dimension) – это множество однотипных данных, образующих одну из граней гиперкуба. Примерами наиболее часто используемых временных измерений являются Дни, Месяцы, Кварталы и годы. В качестве географических измерений широко употребляются Города, Районы, Регионы и Страны. В многомерной модели данных измерения играют роль индексов, служащих для идентификации конкретных значений в ячейках гиперкуба.

Ячейка (Cell) или показатель - это поле, значение которого однозначно определяется фиксированным набором измерений. Тип поля чаще всего определен как цифровой. В зависимости от того, как формируются значения некоторой ячейки, обычно она может быть переменной (значения изменяются и могут быть загружены из внешнего источника данных или сформированы программно) либо формулой (значения, подобно формульным ячейкам электронной таблицы, вычисляются по заранее заданным формулам).

В примере на рис. 2.8 каждое значение ячейки **Объем продаж** однозначно определяется комбинацией временного измерения (Месяц продаж) и модели автомобиля. Пример трехмерной модели данных приведен на рис. 2.8.



Измерения:

Время (год) – 2010, 2011, 2012.

Менеджер – Петров, Смирнов, Яковлев.

Модель – «Opel», «Lada», «KIA».

Показатель: Объем продаж.

Рис. 2.8. Пример трехмерной модели

В существующих МСУБД используются два основных варианта (схемы) организации данных: гиперкубическая и поликубическая.

В полукубической схеме предполагается, что в СУБД может быть определено несколько гиперкубов с различной размерностью и с различными измерениями в качестве граней. Примером системы, поддерживающей поликубический вариант БД, является сервер Oracle Express Server.

В случае гиперкубической схемы предполагается, что все показатели определяются одним и тем же набором измерений. Это означает, что при наличии нескольких гиперкубов БД все они имеют одинаковую размерность и совпадающие измерения. Очевидно, в некоторых случаях информация в БД может быть избыточной (если требовать обязательное заполнение ячеек).

В случае многомерной модели данных применяется ряд специальных операций, к которым относятся: формирование «среза», «вращение», агрегация и детализация.

«Срез» (Slice) представляет собой подмножество гиперкуба, полученное в результате одного или нескольких измерений. Формирование «срезов» выполняется для ограничения используемых пользователем значений, так как все значения гиперкуба практически никогда одновременно не используются. Например, если ограничить значения измерения Модель автомобиля в гиперкубе (рис.2.8) маркой «Lada», то получится двухмерная таблица продаж этой марки автомобиля различными менеджерами по годам.

Операция «вращение» (Rotate) применяется при двумерном представлении данных. Суть ее заключается в изменении порядка измерений при визуальном представлении данных. Так, «вращение» двумерной таблицы, показанной на рис.2.8, приведет к изменению ее вида таким образом, что по оси X будет марка автомобиля, а по оси Y – время.

Операцию «вращения» можно обобщить и на многомерный случай, если под ней понимать процедуру изменения порядка следования измерений. В простейшем случае, это может быть взаимная перестановка двух произвольных измерений.

Операции «агрегация» (Drill Up) и «детализация» (Drill Down) означают соответственно переход к более общему или к более детальному представлению информации пользователю из гиперкуба.

Для иллюстрации смысла операции «агрегация» предположим, что у нас имеется гиперкуб, в котором помимо измерений гиперкуба, приведенного на рис. 2.8, имеются еще измере-

ния: Подразделение, Регион, Фирма, Страна. Заметим, что в этом случае в гиперкубе существует иерархия (снизу вверх) отношений между измерениями: Менеджер, Подразделение, Регион, Фирма, Страна.

Пусть в описанном гиперкубе определено, насколько успешно в 2000 году менеджер Петров продавал автомобили «Lada» и «KIA». Тогда, поднимаясь на уровень выше по иерархии, с помощью операции «агрегация» можно выяснить, как выглядит соотношение продаж этих же моделей на уровне подразделения, где работает Петров.

Основным достоинством многомерной модели данных является удобство и эффективность аналитической обработки больших объемов данных, связанных со временем. При организации обработки аналогичных данных на основе реляционной модели происходит нелинейный рост трудоемкости операций в зависимости от размерности БД и существенное увеличение затрат оперативной памяти на индексацию.

Недостатком многомерной модели данных является ее громоздкость при решении простейших задач оперативной обработки информации.

Примерами систем, поддерживающими многомерные модели данных, являются Essbase (Arbor Software), Media Multimatrix (Speedware), Oracle Express Server (Oracle), Cache (Inter-System). Некоторые программные продукты, например Media/MR (Speedware), позволяют одновременно работать с многомерными и с реляционными БД. В СУБД Oracle, в которой внутренней моделью данных является многомерная модель, реализованы три способа доступа к данным: прямой (на уровне узлов многомерных матриц), объектный и реляционный.

2.6. Объектно-ориентированная модель

В сфере моделирования данных наиболее активно развиваются **объектно-ориентированные базы данных**. Предметная область моделируется как множество классов взаимодейст-

вующих объектов. Каждый объект характеризуется набором свойств, которые являются его характеристиками, и набором методов работы с этим объектом. Работать с объектом можно только с использованием его методов. Атрибуты объекта могут принимать определенное множество допустимых значений, набор конкретных значений атрибутов объекта определяет его состояние. Используя методы работы с объектом, можно изменять значение его атрибутов и тем самым изменять состояние самого объекта. Множество объектов с одним и тем же набором атрибутов и методов образует класс объектов. Объект должен принадлежать только одному классу (если не учитывать возможности наследования). Допускается наличие примитивных предопределенных классов, объекты-экземпляры которых не имеют атрибутов: целые, строки и т.д. Класс, объекты которого могут служить значениями атрибута объектов другого класса, называется доменом этого атрибута.

Одной из перспективных черт объектно-ориентированных моделей данных является принцип наследования. Допускается порождение нового класса на основе уже существующего класса, и этот процесс называется наследованием. В этом случае новый класс, называемый подклассом существующего класса (суперкласса), наследует все атрибуты и методы суперкласса. В подклассе, кроме того, могут быть определены дополнительные атрибуты и методы. Различаются случаи простого и множественного наследования. В первом случае подкласс может определяться только на основе одного суперкласса, во втором случае суперклассов может быть несколько. Если в языке или системе поддерживается единичное наследование классов, то набор классов образует древовидную иерархию. При поддержании множественного наследования классы связаны в ориентированный граф с корнем, называемый решеткой классов. Объект подкласса считается принадлежащим любому суперклассу этого класса.

Наиболее важным качеством объектно-ориентированных баз данных (ООБД) является учет поведенческого аспекта объ-

ектов. В прикладных информационных системах, основывающихся на БД с традиционной организацией, существовал принципиальный разрыв между структурной и поведенческой частями. Структурная часть системы поддерживалась всем аппаратом БД, ее можно было моделировать, редактировать и т.д., а поведенческая часть создавалась изолированно. В частности, отсутствовали формальный аппарат и системная поддержка совместного моделирования и гарантий согласованности структурной (статической) и поведенческой (динамической) частей. В среде ООБД при проектировании, разработке и сопровождении информационной системы учитываются структурные и поведенческие аспекты. Для этого нужны специальные языки, позволяющие определять объекты и создавать на их основе прикладную систему. Произошло также уточнение толкования классических концепций и некоторое их расширение.

Возникло направление, которое предполагает возможность хранения объектов внутри реляционной БД. Необходимость работы с объектами потребовала расширения стандарта языка SQL. Частично это уже сделано в новом стандарте SQL3.

Многие фирмы-разработчики коммерческих СУБД обратились к объектным технологиям и доработали свои СУБД. Например, фирмы IBM и Oracle добавили объектную надстройку над реляционным ядром системы (продукты DB2 и ORACLE). Фирма Informix приобрела объектно-ориентированную СУБД Illustra и встроила ее в свою СУБД.

Стандартизованная объектно-ориентированная модель описана в рекомендациях стандарта ODMG-3 (Object Database Management Group – группа управления объектно-ориентированными базами данных).

В 1991 году была сформирована группа Object Database Management Group (ODMG), перед которой была поставлена цель построить стандарты для объектно-ориентированных баз данных (ООБД). В 1993 году эта группа предложила стандарт для ООБД, названный ODMG-3, который включал:

- объектную модель Object Data Model (ODM);
- язык определения объектов Object Definition Language (ODL);
- объектный язык запросов Object Query Language (OQL);
- интерфейсы языков программирования (C++ и других).

Реализовать в полном объеме рекомендации ODMG-3 пока не удастся. В настоящее время насчитывается свыше 300 объектно-ориентированных СУБД (ООСУБД). Сведения о некоторых из них приведены в табл. 2.1.

Таблица 2.1

Характеристики некоторых ООСУБД

Фирма – производитель	Название ООСУБД	Средства разработки	Подход к разработке
Objectivity	Objectivity/DB	C, C++, SQL, Java	Расширение объектно-ориентированных библиотек классов
Poet Software	Poet	C, C++, ODBC, Java	
Object Design	Object Store	C, C++, Java	
Ontos Inc.		C++, Java	
Versant Object Technology	Ontos DB, Versant	C++, Java	
Computer Associates	Jasmine	C++, Java	
НППЦ «Интеллект Плюс»	ODB-Jupiter	C++	
O2 Technology	O2	C++, Java	Вставка объектно-ориентированного языка БД в обычный базовый язык
GemStone Inc.	GemStone	C++, Java	Расширение языка (C++) возможностями работы с БД

InterSystems	Cache	Semantic Information Manager, Cache ObjectScript	Новый язык базы данных или модели данных
--------------	-------	--	--

Для иллюстрации ключевых идей рассмотрим несколько упрощенную модель объектно-ориентированной БД.

Структура объектно-ориентированной БД графически представима в виде дерева, узлами которого являются объекты. Свойства объектов описываются некоторым стандартным типом (например, строковым – string) или типом, конструируемым пользователем (определяется как class).

Значением свойства типа string является строка символов. Значение свойства типа class есть объект, являющийся экземпляром соответствующего класса. Каждый объект-экземпляр класса считается потомком объекта, в котором он определен как свойство. Объект-экземпляр класса принадлежит своему классу и имеет одного родителя. Родовые отношения в БД образуют иерархию объектов.

Пример логической структуры объектно-ориентированной БД для библиотеки приведен на рис. 2.9.

Здесь объект типа БИБЛИОТЕКА является родительским для объектов-экземпляров классов АБОНЕНТ, КАТАЛОГ и ВЫДАЧА. Различные объекты типа КНИГА могут иметь одного или разных родителей. Объекты типа КНИГА, имеющие одного и того же родителя, должны различаться по крайней мере инвентарным номером (уникален для каждого экземпляра книги), но имеют одинаковые значения свойств: шифр книги, УДК, название и автор.

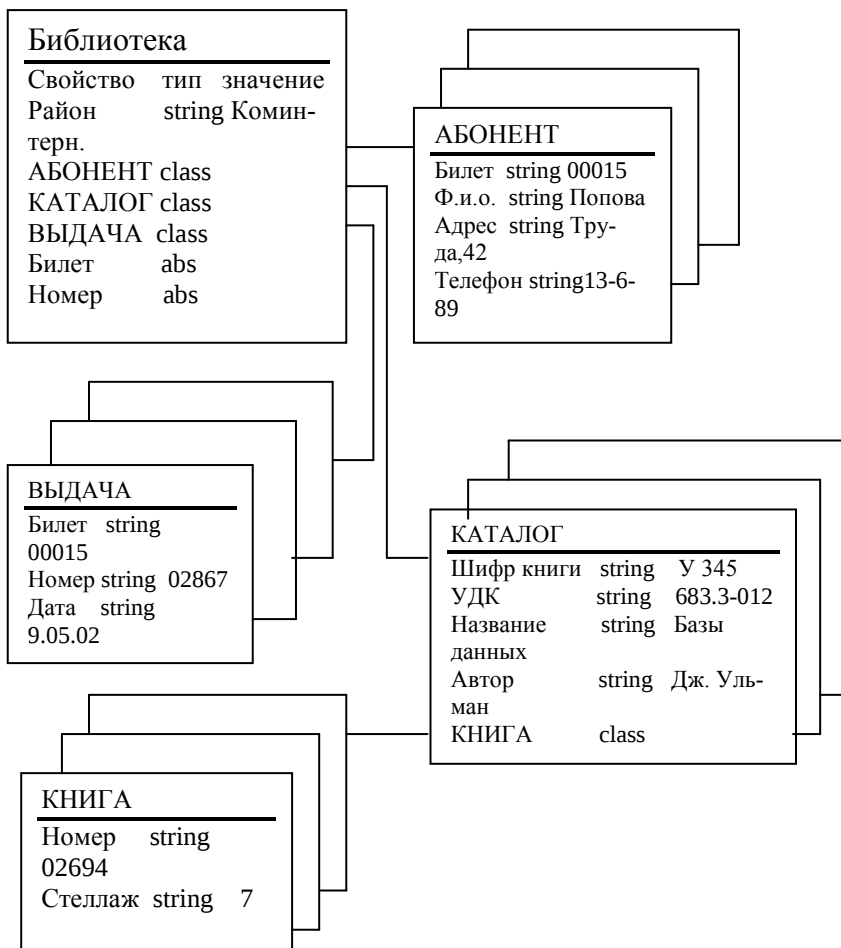


Рис. 2.9. Логическая структура БД библиотеки

Логическая структура объектно-ориентированной БД внешне похожа на структуру иерархической БД. Основное отличие между ними состоит в методах манипулирования данными.

Для выполнения действий над данными в рассматриваемой модели БД применяются логические операции, усиленные

объектно-ориентированными механизмами инкапсуляции, наследования и полиморфизма. Ограниченно могут применяться операции, подобные командам SQL (например, для создания БД).

Создание и модификация базы данных сопровождается автоматическим формированием и последующей корректировкой индексов (индексных таблиц), содержащих информацию для быстрого поиска данных.

Рассмотрим кратко понятия инкапсуляции, наследования и полиморфизма применительно к объектно-ориентированной модели БД.

Инкапсуляция – выделение класса с доступом к нему через свойства и методы. Инкапсуляция ограничивает область видимости имени свойства пределами того объекта, в котором оно определено. Так, если в объект типа КАТАЛОГ добавить свойство, задающее телефон автора книги и имеющее название телефон, то мы получим одноименные свойства у объектов АБОНЕНТ и КАТАЛОГ. Смысл такого свойства будет определяться тем объектом, в котором оно инкапсулировано.

Наследование – трансформация класса путем изменения свойств и методов с помощью методов, называемых конструктором и деструктором. Все классы строятся по иерархическому принципу с происхождением от некоторого исходного класса.

Наследование распространяет область видимости свойства на всех потомков объекта. Так, всем объектам типа КНИГА, являющимся потомками объекта типа КАТАЛОГ, можно приписать свойства объекта-родителя: шифр книги, УДК, название, автор. Если необходимо расширить действие механизма наследования на объекты, не являющиеся непосредственно родственниками (например, между двумя потомками одного родителя), то в их общем предке определяется абстрактное свойство типа abs. Так, определение абстрактных свойств билет и номер в объекте БИБЛИОТЕКА приводит к наследованию этих свойств всеми дочерними объектами АБОНЕНТ, КНИГА и ВЫДАЧА. Не случайно поэтому значения свойства билет клас-

сов АБОНЕНТ и ВЫДАЧА, показанных на рисунке, будут одинаковыми – 00015.

Полиморфизм позволяет использовать метод с одним и тем же именем как в базовом, так и в производных классах. Полиморфизм в объектно-ориентированных языках программирования означает способность одного и того же программного кода работать с разнотипными данными. Другими словами, он означает допустимость в объектах разных типов иметь методы (процедуры или функции) с одинаковыми именами. Во время выполнения объектной программы одни и те же методы оперируют с разными объектами в зависимости от типа аргумента. Применительно к нашей объектно-ориентированной базе данных полиморфизм означает, что объекты класса КНИГА, имеющие разных родителей из класса КАТАЛОГ, могут иметь разный набор свойств. Следовательно, программы работы с объектами класса КНИГА могут содержать полиморфный код.

Поиск в объектно-ориентированной БД состоит в выяснении сходства между объектом, задаваемым пользователем, и объектами, хранящимися в БД. Определяемый пользователем объект, называемый объектом-целью (свойство объекта имеет тип goal), в общем случае может представлять собой подмножество всей хранимой в БД иерархии объектов. Объект-цель, а также результат выполнения запроса могут храниться в самой базе.

Приведем **основные характеристики ООБД**, базирующиеся на объектно-ориентированном подходе.

1. В качестве значения столбца может быть использован произвольный кортеж. Иными словами, столбец может быть другой таблицей. Таким образом, создается возможность реализации таблиц с нелинейной структурой.

2. Процедура манипуляции данными позволяет присоединять процедуры, определенные значениями столбцов.

3. Данные столбцов могут наследоваться.

4. Элементами отношений могут быть не только отдельные элементы, но и множества.

5. Формируются классы данных, которые организуют столбцы в иерархию.

Базовым языком ООБД чаще всего является C++. Для работы с ООБД разрабатывается новый вариант языка запросов SQL.

В технологии разработки ООБД конкурирует два направления:

- Distributed Object Linking and Embedding (OLE) фирмы Microsoft;

- Common Object Request Broker Architecture (CORBA) группы OBDMG, поддерживаемая фирмами IBM, Novell, DEC.

Таким образом, чтобы воспользоваться объектно-ориентированным подходом в построении БД, необходимо:

- провести инкапсуляцию данных, т.е. выделить классы и объекты;

- определить возможные виды структуры реализуемых таблиц;

- создать наследование классов данных;

- обеспечить полиморфизм.

Основным достоинством объектно-ориентированной модели данных в сравнении с реляционной является возможность отображения информации о сложных взаимосвязях объектов. Объектно-ориентированная модель данных позволяет идентифицировать отдельную запись базы данных и определять функции их обработки.

Недостатками объектно-ориентированной модели являются высокая понятийная сложность, неудобство обработки данных и низкая скорость выполнения запросов.

В 90-е годы существовали экспериментальные прототипы объектно-ориентированных систем управления базами данных. В настоящее время такие системы получили широкое распространение, в частности, к ним относятся следующие СУБД: POET (POET Software), Jasmine (Computer Associates), Versant

(Versant Technologies), Q2 (Ardent Software), ODB-Jupiter (научно-производственный центр “Интелтек Плюс”), а также Iris, Orion, Postgres.

2.7. Объектно-реляционная модель данных

Промежуточной моделью данных между реляционными и объектно-ориентированными моделями данных является объектно-реляционная модель данных (ОРБД).

Различают две разновидности ОРБД – **гибридные** и **расширенные**.

В **гибридных** ОРБД интерфейс пользователя и алгоритм приложения выполнены с применением объектно-ориентированного подхода, а БД является реляционной. Примерами могут служить СУБД Paradox и InterBase, используемые для создания и использования базы данных в приложениях, разработанных в среде Delphi. В каком-то смысле гибридной можно считать СУБД Access, которая используется для создания базы данных и приложения и использует встроенный язык программирования Visual Basic for Application (VBA).

В **расширенных** (постреляционных) ОРБД предполагается объектно-ориентированное построение собственно базы данных путем использования известных и введения новых типов данных, связанных между собой. Эта связь реализуется через методы, описываемые с помощью триггеров и хранимых процедур. В расширенной объектно-реляционной модели допускается неатомарность данных, записываемых в поле. В полях может располагаться другая таблица или массив данных. К подобным СУБД относятся Informix Universal Server, Oracle 8, UniSQL. Данные СУБД используют язык запросов SQL3.

Постреляционными называют БД, в которых частично проведена денормализация данных и частично допускается неатомарность записей.

В настоящий период широкое распространение получила гибридная разновидность ОРБД.

К достоинствам ОРБД следует отнести следующие;

- повторное использование компонентов, имеющих заданный набор свойств и методов;
- частичное или полное отсутствие нормализации таблиц;
- возможность повтора данных в поле.

К недостаткам ОРБД можно отнести следующее:

- усложнение структуры базы данных;
- сложность построения абстрактных типов данных и методов, связывающих типы в иерархию.

3. РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ

3.1. Основные определения

3.1.1. Определение отношения, домена, кортежа, реляционной базы данных, ключей

Реляционная модель данных была предложена Е. Коддом в 1970 году. В основе реляционной модели данных лежит понятие отношения.

Математически отношение определяется следующим образом. Пусть даны n множеств $D_1, D_2, \dots, D_n$. Тогда R есть отношение над этими множествами, если R есть множество упорядоченных кортежей длины n вида $(d_1, d_2, \dots, d_n)$, где d_1 – элемент из D_1 , d_2 – элемент из D_2 , d_n – элемент из D_n . $D_1, D_2, \dots, D_n$ называют доменами отношения R . Заметим, что данное определение эквивалентно определению декартова произведения множеств $D_1, D_2, \dots, D_n$.

Дадим определение отношения с точки зрения теории обработки данных. **Отношение** – подмножество декартова произведения одного или более доменов. **Домен** – множество возможных значений конкретного атрибута. **Атрибут** – свойство объекта, явления или процесса. Примеры атрибутов: фамилия, имя, отчество, дата рождения. **Кортеж** – элемент отношения, это отображение имен атрибутов в значения, взятые из соответствующих доменов. Конечное множество кортежей образует

отношение. Если отношение создается из n доменов, то каждый кортеж имеет n компонент.

Поясним данные определения на примерах.

Пример 1. Пусть имеется два домена:

$D1 = (0,1)$; $D2 = (a,b,c)$.

Построим декартово произведение доменов $D1, D2$:

$D1 \times D2 = \{(0,a),(0,b),(0,c),(1,a),(1,b),(1,c)\}$.

В качестве отношения, построенного на доменах $D1, D2$, можно выбрать, например, следующее:

$R = \{(0,a),(0,c),(1,b),(1,c)\}$.

Отношение R состоит из четырех кортежей, в каждом кортеже по два элемента, первый выбирают из домена $D1$, второй – из домена $D2$.

Пример 2. Пусть имеется четыре домена:

$D1$ – множество целых чисел, например, множество номеров деталей (101, 34, 23, 109, 147).

$D2$ – множество символьных строк, например, множество названий деталей (втулка, кронштейн, скоба, муфта, болт).

$D3$ – множество символьных строк, например, множество названий видов обработки (холодная штамповка, металлическое литье, литье из пластмасс, механическая обработка).

$D4$ – множество вещественных чисел, например, множество весов деталей (45.8, 6.9, 123, 69.3, 5.2, 2.34).

В качестве отношения, построенного на доменах $D1, D2, D3, D4$, можно выбрать, например, следующее:

$R = \{(34, \text{втулка, литье из пластмасс, 69.3}), (23, \text{кронштейн, холодная штамповка, 45.8}), (101, \text{болт, механическая обработка, 5.2})\}$.

Отношение R состоит из трех кортежей, в каждом кортеже по четыре элемента.

Удобно представить отношение как таблицу, где каждая строка есть кортеж, содержащий данные о конкретном объекте, явлении или процессе. Каждый столбец таблицы – это домен, содержащий возможные значения одного из свойств объекта, процесса или явления.

Например:

Детали

Номер детали	Название детали	Вид обработки	Вес
401000	Втулка	Литье из пластмасс	69.3
409857	Кронштейн	Холодная штамповка	45.8
409900	Стойка	Механическая обработка	5.2
409870	Скоба	Холодная штамповка	0.2
409859	Шайба	Токарная обработка	0.3
409887	Шасси	Механическая обработка	1.4

Следующие наборы терминов эквивалентны:

отношение, таблица, файл;

кортеж, строка, запись;

атрибут, ячейка, поле.

Поименованный список имен атрибутов отношения называют **схемой отношения**. Пример схемы:

Детали (Номер детали, Название детали, Вид обработки, Вес).

Ключевой атрибут в схеме отношения подчеркивают.

Совокупность схем отношений, используемых для представления информации, называется **схемой реляционной базы данных**.

Число столбцов в отношении называют **степенью**. Текущее число кортежей в отношении называют мощностью. Степень отношения обычно не изменяется после создания отношения, но мощность будет колебаться по мере добавления новых и удаления старых кортежей.

Реляционная база данных — это совокупность отношений, содержащих всю информацию, которая должна храниться в базе данных.

Пример фрагмента базы данных.

Отношение 1. Радиоэлементы (транзисторы)

Тип прибора	Материал	Структура	Технология	Мощность коллектора
ГТ109Г	Ge	р-п-р	С	30
ГТ115Б	Ge	р-п-р	С	50
КТ208А	Si	р-п-р	ПЭ	200

Отношение 2. Склад

Номер стеллажа	Тип прибора	Количество
10	ГТ109Г	100
11	ГТ109Г	50
10	КТ208А	12

Каждое отношение имеет ключ. **Ключ (первичный ключ, ключ отношения, ключевой атрибут)** – это атрибут или группа атрибутов, которые позволяют однозначно идентифицировать кортеж в отношении.

Если **ключ составной** (состоит из двух и более атрибутов), то он должен быть **минимальным**. Это значит, что если один произвольный атрибут исключить из составного ключа, оставшихся атрибутов будет недостаточно для однозначной идентификации отдельных кортежей.

Значения ключа в отношении (таблице) должны быть уникальными, то есть не должны существовать два или более кортежа (записи) с одинаковым значением ключа. Если в отношении нет полей, значения в которых уникальны, для создания ключа вводят обычно дополнительное числовое поле, содержащее порядковые номера записей.

В отношении "Радиоэлементы (транзисторы)" ключом является Тип прибора, в отношении "Склад" - Номер стеллажа, Тип прибора.

Возможны случаи, когда отношение имеет несколько комбинаций атрибутов, каждая из которых однозначно определяет все кортежи отношения. Все эти комбинации атрибутов

являются **возможными ключами** отношения. Любой из возможных ключей может быть выбран как первичный.

Ключи обычно используют для достижения следующих целей: исключения дублирования значений в ключевых атрибутах; упорядочения кортежей; ускорения работы с кортежами отношения; организации связывания таблиц.

Пусть в отношении R1 имеется не ключевой атрибут A, значения которого являются значениями ключевого атрибута B другого отношения R2. Тогда говорят, что атрибут A отношения R1 (атрибут B отношения R2) есть внешний ключ. С помощью внешних ключей устанавливаются связи между отношениями.

3.1.2. Классы отношений

Отношения реляционной базы данных в зависимости от содержания подразделяются на два класса: объектные отношения и связные отношения.

Объектные отношения хранят данные о группах однородных объектов, явлений или процессов, имеющих однотипные характеристики. В объектном отношении ключ называют первичным, или просто ключом отношения.

Связное отношение хранит данные о связях между объектными отношениями. Связное отношение содержит ключи связанных объектных отношений и данные, количественно или качественно характеризующие связь. Ключи связных отношений называют внешними ключами, поскольку они являются первичными ключами других отношений. Реляционная модель накладывает на внешние ключи ограничение, называемое ссылочной целостностью. Это означает, что каждому значению внешнего ключа должен соответствовать кортеж объектного отношения. Без этого возможна ситуация, когда внешний ключ ссылается на объект, о котором ничего не известно.

Рассмотрим пример объектных и связных отношений.

Объектное отношение "Детали"

Номер детали	Название детали
401000	Втулка
409857	Кронштейн

Объектное отношение "Материалы"

Код материала	Марка	Вид материала
181508	АД1Н	Лист
181440	АМГ2М	Лист

Связное отношение "Технологический процесс"

Номер детали	Код материала	Норма расхода материала
401000	181508	23.78
409857	181440	112.6

В отношении "Детали" первичный ключ – номер детали. В отношении "Материал" первичный ключ – код материала. В отношении "Технологический процесс" внешний ключ – номер детали, код материала. Атрибут "Норма расхода материала на деталь" – количественная характеристика связи между деталью и материалом.

Поскольку не всякой таблице можно поставить в соответствие отношение, приведем условия, выполнение которых позволяет считать таблицу отношением.

1. Все строки таблицы должны быть уникальны, т.е. не может быть строк с одинаковыми первичными ключами.
2. Имена столбцов таблицы должны быть различны, а значения их простыми, т.е. недопустима группа значений в одном столбце одной строки.
3. Все строки одной таблицы должны иметь одинаковую структуру с соответствующими именами и типами столбцов.

4. Порядок размещения строк в таблице может быть произвольным.

3.1.3. Индексирование

Определение ключа для таблицы означает автоматическую сортировку записей, контроль отсутствия повторений значений в ключевых полях записей и повышение скорости выполнения операций поиска в таблице. Для реализации этих функций в СУБД применяют индексирование. **Индекс** – это средство ускорения операции поиска записей в таблице, а, следовательно, и других операций, использующих поиск: извлечение, модификация, сортировка и т.д. Таблицу, для которой используется индекс, называют индексированной. Ключевые поля таблицы во многих СУБД, как правило, индексируются автоматически. Индексы, созданные для ключей, называют **первичными индексами**.

Индексы, создаваемые пользователем для неключевых полей, иногда называют **вторичными (пользовательскими) индексами**. Введение таких индексов не изменяет физического расположения записей таблицы, но влияет на последовательность просмотра записей.

Главная причина повышения скорости выполнения различных операций в индексированных таблицах состоит в том, что основная часть работы производится с небольшими индексными файлами, а не с самими таблицами. Наибольший эффект повышения производительности работы с индексированными таблицами достигается для значительных по объему таблиц. Индексирование требует небольшого дополнительного места на диске и незначительных затрат процессора на изменение индексов в процессе работы.

Индекс или индексная таблица обычно содержит поле (поля), по которому строится индекс, и порядковый номер записи в основной таблице, которая содержит поле с данным ин-

дексным значением. Данные в индексной таблице упорядочены по полю, по которому строится индекс.

Пусть имеется таблица со следующей схемой данных:

Студенты (Номер зачетки, Фамилия, Имя, Отчество, Дата рождения).

Фрагмент таблицы с исходными данными следующий:

06023	Петров Сергей Алексеевич	12.09.89
-------	--------------------------	----------

06010	Волков Максим Михайлович	30.09.89
-------	--------------------------	----------

06002	Антипов Петр Андреевич	09.01.89
-------	------------------------	----------

06018	Орлов Алексей Викторович	23.06.89
-------	--------------------------	----------

Вид индексной таблицы, построенной по номеру зачетки следующий:

06002	3
-------	---

06010	2
-------	---

06018	4
-------	---

06023	1
-------	---

Вид индексной таблицы, построенной по фамилии следующий:

Антипов	3
---------	---

Волков	2
--------	---

Орлов	4
-------	---

Петров	1
--------	---

Вид индексной таблицы, построенной по дате рождения:

09.01.89	3
----------	---

23.06.89	4
----------	---

12.09.89	1
----------	---

30.09.89	2
----------	---

3.1.4. Связи между отношениями (таблицами)

Обычно база данных представляет собой набор связанных таблиц. *Связывание таблиц дает следующие преимущества:*

многие СУБД при связывании таблиц автоматически выполняют контроль целостности вводимых в базу данных в соот-

ветствии с установленными связями, что повышает достоверность хранимой в БД информации;

облегчается доступ к данным. Связывание таблиц при выполнении таких операций, как поиск, просмотр, редактирование, выборка и подготовка отчетов с использованием информации из разных таблиц уменьшает количество явных обращений к таблицам данных и число манипуляций в каждой из них.

Существует несколько разновидностей связей между отношениями. Связанные отношения часто взаимодействуют по принципу главная и подчиненная таблицы. Главную таблицу можно еще называть родительской, а подчиненную – дочерней. Одна и та же таблица может быть главной по отношению к одной таблице базы данных и дочерней по отношению к другой.

Связь «один-ко-многим» означает, что одной записи в родительской таблице может соответствовать несколько записей (в том числе и одна) в дочерней таблице. В родительской таблице могут быть записи, для которых в данный момент нет соответствующих записей в дочерней таблице. Различают также жесткую связь «один-ко-многим», когда каждой записи в родительской таблице должны соответствовать записи в дочерней таблице.

Связь «один-ко-многим» является самой распространенной для реляционных баз данных.

Примеры связи.

Таблицы «Группы» и «Студенты» могут быть связаны связью «один-ко-многим» по полю «Шифр группы». Данная связь будет означать, что одна запись о группе из таблицы «Группы» может быть связана с несколькими записями из таблицы «Студенты», содержащими информацию о студентах данной группы. В связи Группы – Студенты таблица «Группы» является главной, а таблица «Студенты» - подчиненной.

Таблицы «Студенты» и «Экзамены» могут быть связаны связью «один-ко-многим» по полю «Номер Зачетки». Данная связь будет означать, что одна запись о студенте из таблицы «Студенты» может быть связана с несколькими записями о сда-

че экзаменов данным студентом в таблице «Экзамены». В связи Студенты – Экзамены таблица «Студенты» является главной, а таблица «Экзамены» - подчиненной.

Связь «один-к-одному» имеет место, когда одной записи в родительской таблице соответствует только одна запись в дочерней таблице. Данная связь встречается редко и означает, что информация из двух таблиц могла бы быть объединена в одну. Наличие двух таблиц говорит о желании разделить основную и второстепенную информацию на два отношения.

Например, информация о студентах может быть разделена на две таблицы «Студенты» и «Дополнительные сведения», которые будут связаны связью «один-к-одному» по полю «Номер зачетки». Связь «один-к-одному» приводит к тому, что для чтения связанной информации в нескольких таблицах приходится производить несколько операций чтения, что замедляет получение нужной информации. Связь «один-к-одному» может быть жесткой и нежесткой.

Третий вид связи – **связь «многие-ко-многим»**. Данный вид связи означает, что несколько записей одной таблицы связаны с несколькими записями другой таблицы и наоборот. Например: между таблицами «Учебные группы и дисциплины» и «Преподаватели» может существовать связь «многие-ко-многим». Это означает, что каждый преподаватель может вести несколько предметов и, в то же время, один и тот же предмет могут вести несколько преподавателей.

Некоторые СУБД не поддерживают связи «многие-ко-многим» на уровне ссылочной целостности, хотя и позволяют реализовывать ее в таблицах неявным образом. Считается, что базу данных всегда можно перестроить так, чтобы любая связь «многие-ко-многим» была заменена на одну или более связей «один-ко-многим».

3.1.5. Обеспечение целостности данных

Одним из основополагающих понятий в технологии баз данных является понятие целостности. В общем случае это по-

нятие, прежде всего, связано с тем, что база данных отражает в информационном виде некоторый объект реального мира или совокупность взаимосвязанных объектов реального мира. В реляционной модели объекты реального мира представлены в виде совокупности взаимосвязанных отношений. Под **целостностью** будем понимать соответствие информационной модели предметной области, хранимой в базе данных, объектам реального мира и их взаимосвязям в каждый момент времени. Любое изменение в предметной области, значимое для построенной модели, должно отражаться в базе данных, и при этом должна сохраняться однозначная интерпретация информационной модели в терминах предметной области.

Поддержка целостности в реляционной модели данных в ее классическом понимании включает в себя 3 аспекта.

Во-первых, это поддержка **структурной целостности**, которая трактуется как то, что реляционная СУБД должна допускать работу только с однородными структурами данных типа «реляционное отношение». При этом понятие «реляционное отношение» должно удовлетворять всем ограничениям, накладываемым на него в классической теории реляционной БД (отсутствие дубликатов кортежей, обязательное наличие первичного ключа, отсутствие понятия упорядоченности кортежей).

В дополнение к структурной целостности необходимо рассмотреть проблему неопределенных Null значений. Неопределенное значение интерпретируется в реляционной модели как значение, неизвестное на данный момент времени. Это значение при появлении дополнительной информации в любой момент времени может быть заменено на конкретное значение. При сравнении неопределенных значений не действуют стандартные правила сравнения: одно неопределенное значение никогда не считается равным другому неопределенному значению. Для выявления равенства значения некоторого атрибута неопределенному значению применяются специальные стандартные предикаты:

<имя атрибута> IS NULL и <имя атрибута> IS NOT NULL.

Если в данном кортеже (в данной строке) указанный атрибут имеет неопределенное значение, то предикат IS NULL принимает значение TRUE (Истина), а предикат IS NOT NULL – FALSE (Ложь), в противном случае предикат IS NULL принимает значение FALSE, а предикат IS NOT NULL принимает значение TRUE.

Введение Null значений вызвало необходимость модификации классической двузначной логики и превращения ее в трехзначную.

Во-вторых, это поддержка **языковой целостности**, которая состоит в том, что реляционная СУБД должна обеспечивать языки описания и манипулирования данными не ниже стандарта SQL. Не должны быть доступны иные низкоуровневые средства манипулирования данными, не соответствующие стандарту.

В-третьих, это поддержка **ссылочной целостности** (Declarative Referential Integrity, DRI).

Ссылочная целостность – это совокупность связей между отдельными таблицами во всей базе данных. Нарушение хотя бы одной такой связи делает информацию в базе данных недостоверной. СУБД обычно блокирует действия, которые нарушают целостность связей между таблицами, т.е. нарушают ссылочную целостность. Обеспечение ссылочной целостности означает, что СУБД при корректировке базы данных обеспечивает для связанных таблиц контроль за соблюдением следующих правил:

- в подчиненную таблицу не может быть добавлена запись с несуществующим в главной таблице значением ключа связи;

- в главной таблице нельзя удалить запись, если не удалены связанные с ней записи в подчиненной таблице;

- изменение значений ключа связи в записи главной таблицы невозможно, если в подчиненной таблице имеются связанные с ней записи.

При попытке пользователя нарушить эти условия в операциях добавления и удаления записей или обновления ключевых

данных в связанных таблицах СУБД должна выводить сообщения об ошибке и не допускать выполнения этих операций.

Чтобы предотвратить потерю ссылочной целостности, используется механизм каскадных изменений. Он состоит в обеспечении следующих действий:

при изменении поля связи в записи родительской таблицы следует синхронно изменить значения полей связи в соответствующих записях дочерней таблицы;

при удалении записи в родительской таблице следует удалить соответствующие записи в дочерней таблице.

Структурная, языковая и ссылочная целостности не определяют семантику БД, не касаются содержания базы данных, поэтому вводится понятие **семантической поддержки целостности**.

Семантическая поддержка может быть обеспечена двумя путями: декларативным и процедурным путем. Декларативный путь связан с наличием механизмов в рамках СУБД, обеспечивающих проверку и выполнение ряда декларативно заданных правил-ограничений, называемых чаще всего «бизнес-правилами» (Business Rules) или декларативными ограничениями целостности.

Выделяются следующие виды декларативных ограничений целостности:

ограничения целостности атрибута: значение по умолчанию, задание обязательности или необязательности значений (Null), задание условий на значения атрибутов.

Задание значения по умолчанию означает, что каждый раз при вводе новой строки в отношение, при отсутствии данных в указанном столбце этому атрибуту присваивается именно значение по умолчанию. Например, при вводе новых записей в поле год издания необходимо ввести значение текущего года. Для MS Access это выражение будет иметь вид:

YEAR(NOW())

здесь NOW() – функция, возвращающая значение текущей даты, YEAR(data) – функция, возвращающая значение года для даты, указанной в качестве параметра.

Другой пример, в качестве условия на значение для года издания надо задать выражение, которое будет проверять попадание года издания в интервал от 1960 года до текущего года. Для MS Access это выражение будет выглядеть следующим образом:

Between 1960 AND YEAR(NOW())

В СУБД MS SQL Server значение по умолчанию записывается в качестве «бизнес-правила». В этом случае будет использоваться выражение, в котором явным образом должно быть указано имя соответствующего столбца, например:

YEAR_PUBL>=1960 AND YEAR_PUB<= YEAR(GETDATE())

Здесь GETDATE() – функция MS SQL Server, возвращающая значение текущей даты, YEAR_PUB – имя столбца, соответствующего году издания.

Ограничения целостности, задаваемые на уровне доменов. Эти ограничения удобны, если в базе данных присутствуют несколько столбцов разных отношений, которые принимают значения из одного и того же множества допустимых значений. Некоторые СУБД разрешают определять отдельно домены, задавать тип данных для каждого домена и задавать соответственно ограничения в виде бизнес-правил для доменов. В этом случае для атрибутов задается принадлежность к тому или иному домену. Иногда доменная структура выражена неявно. Так, например, в MS SQL Server вместо понятия домена вводится понятие типа данных, определенных пользователем, но смысл этого типа данных фактически эквивалентен смыслу домена. Удобно задать ограничение на значение на уровне домена, тогда оно автоматически будет выполняться для всех атри-

бутов, принимающих значения из этого домена. Если меняется ограничение, то его замена проводится один раз на уровне домена, а все атрибуты, которые принимают значения из этого домена, будут автоматически работать по новому правилу.

Ограничения целостности, задаваемые на уровне отношения. Некоторые семантические правила невозможно преобразовать в выражения, которые будут применимы только к одному столбцу. Например, при создании отношения Читатели потребовать наличия по крайней мере одного телефонного номера (домашнего или рабочего) для быстрой связи с читателем. Для MS Access или MS SQL Server соответствующее выражение будет следующим:

HOME_PHON IS NOT NULL OR WORK_PHON IS NOT NULL

Ограничения целостности, задаваемые на уровне связи между отношениями: задание обязательности связи, принципов каскадного удаления и каскадного обновления данных, задание поддержки ограничений по мощности связи. Эти виды ограничений могут быть выражены заданием обязательности или необязательности значений внешних ключей во взаимосвязанных отношениях.

Декларативные ограничения целостности относятся к ограничениям, которые являются немедленно проверяемыми. Есть ограничения целостности, которые являются откладываемыми. Эти ограничения целостности поддерживаются механизмом транзакций и триггеров.

3.2. Операции реляционной алгебры

3.2.1. Основные понятия

В реляционных СУБД для выполнения операций над отношениями используются две группы языков, имеющие в качестве своей математической основы теоретические языки запросов, предложенные Э. Коддом:

- реляционная алгебра;
- реляционное исчисление.

В *реляционной алгебре* операнды и результаты всех действий являются отношениями. Языки реляционной алгебры являются процедурными, так как отношение, являющееся результатом запроса к реляционной базе данных, вычисляется при выполнении последовательности реляционных операторов, применяемых к отношениям. Операторы состоят из операндов, в роли которых выступают отношения, и реляционных операций. Результатом реляционной операции является отношение.

Языки *исчислений*, в отличие от реляционной алгебры, являются непроцедурными (описательными или декларативными) и позволяют выражать запросы с помощью предиката первого порядка (высказывания в виде функции), которому должны удовлетворять кортежи или домены отношений. Запрос к БД, выполненный с использованием подобного языка, содержит лишь информацию о желаемом результате. Для этих языков характерно наличие набора правил для записи запросов. В частности, к языкам этой группы относится SQL.

Реляционная алгебра, предложенная Коддом, включает в себя следующие основные операции: объединение, разность (вычитание), пересечение, декартово (прямое) произведение (или произведение), выборка (селекция, ограничение), проекция, деление и соединение.

По справедливому замечанию Дейта, реляционная алгебра Кодда обладает несколькими недостатками. Во-первых, восемь перечисленных операций по охвату своих функций, с одной стороны, избыточны, так как минимально необходимый набор составляют пять операций: объединение, вычитание, произведение, проекция и выборка. Три других операции (пересечение, соединение и деление) можно определить через пять минимально необходимых. Так, например, соединение – это проекция выборки произведения.

Во-вторых, этих восьми операций недостаточно для построения реальной СУБД на принципах реляционной алгебры. Требуются расширения, включающие операции: переименования атрибутов, образования новых вычисляемых атрибутов,

вычисления итоговых функций, построения сложных алгебраических выражений, присвоения, сравнения и т.д.

Операции реляционной алгебры Кодда можно разделить на две группы: базовые теоретико-множественные и специальные реляционные. Первая группа операций включает в себя классические операции теории множеств: объединение, разность, пересечение и произведение. Вторая группа представляет собой развитие обычных теоретико-множественных операций в направлении к реальным задачам манипулирования данными. В ее состав входят следующие операции: проекция, селекция, деление и соединение.

Операции реляционной алгебры могут выполняться над одним отношением (например, проекция) или над двумя отношениями (например, объединение). В первом случае операция называется унарной, а во втором – бинарной. Отношения, участвующие в бинарной операции, должны быть совместимы по структуре.

Совместимость структур отношений означает совместимость имен атрибутов и типов соответствующих доменов. Частным случаем совместимости является идентичность (совпадение). Для устранения конфликтов имен атрибутов в исходных отношениях (когда совпадение имен недопустимо), а также для построения произвольных имен атрибутов результирующего отношения применяется операция переименования атрибутов. Структура результирующего отношения по определенным правилам наследует свойства структур исходных отношений. В рассматриваемых примерах будем считать, что заголовки исходных отношений идентичны.

3.2.2. Базовые теоретико-множественные операции реляционной алгебры

Объединением двух совместимых отношений R_1 и R_2 одинаковой размерности ($R_1 \cup R_2$) является отношение R_3 , со-

держашее все элементы исходных отношений (с исключением повторений).

В примере применения операции объединения отношения R1 и R2 содержат перечни деталей, изготавливаемых соответственно на первом и втором участках цеха. Отношение R3 содержит общий перечень деталей, изготавливаемых в цехе, то есть характеризует общую номенклатуру деталей цеха.

R1	
Шифр детали	Название детали
003412	Гайка M1
003415	Гайка M2
003477	Болт M1

R2	
Шифр детали	Название детали
003412	Гайка M1
003416	Гайка M3

R3	
Шифр детали	Название детали
003412	Гайка M1
003415	Гайка M2
003416	Гайка M3
003477	Болт M1

Пересечением двух совместимых отношений R1 и R2 одинаковой размерности ($R1 \cap R2$) называется отношение R3, содержащее множество кортежей, принадлежащих одновременно и первому и второму отношениям.

В отношении R4 содержится перечень деталей, которые выпускаются одновременно на двух участках цеха.

R4	
Шифр детали	Название детали
003412	Гайка M1

Разностью двух совместимых отношений $R1$ и $R2$ одинаковой размерности $(R1 \setminus R2)$ называется отношение, содержащее множество кортежей, принадлежащих $R1$ и не принадлежащих $R2$.

Отношение $R5$ содержит перечень деталей, изготавливаемых только на участке1 $(R1 \setminus R2)$, отношение $R6$ содержит перечень деталей, изготавливаемых только на участке 2 $(R6 = R2 \setminus R1)$.

R5	
Шифр детали	Название детали
003415	Гайка M2
003477	Болт M1

R6	
Шифр детали	Название детали
003416	Гайка M3

Следует отметить, что первые две операции, объединение и пересечение, являются коммутативными операциями, то есть результат операции не зависит от порядка аргументов в операции. Операция же разности является принципиально несимметричной, то есть результат операции будет различным для разного порядка аргументов, что и видно из сравнения отношений $R5$ и $R6$.

Четвертой теоретико-множественной операцией является *расширенное декартово произведение*. Эта операция не накладывает никаких дополнительных условий на схемы исходных отношений, поэтому операция расширенного декартова произведения, обозначаемая $R1 \otimes R2$, допустима для любых двух отношений. Введем дополнительное понятие конкатенации, или сцепления кортежей.

Сцеплением или *конкатенацией* кортежей $s = \langle c_1, c_2, \dots, c_n \rangle$ и $q = \langle q_1, q_2, \dots, q_m \rangle$ называется кортеж, полученный добавлением значений второго кортежа в конец первого. Сцепление кортежей s и q обозначается как (s, q) .

$$(s, q) = \langle c_1, c_2, \dots, c_n, q_1, q_2, \dots, q_m \rangle.$$

Здесь n – число элементов в первом кортеже S , m – число элементов во втором кортеже q .

Все предыдущие операции не меняли степени или арности отношений, это следовало из эквивалентности схем отношений. Операция декартова произведения меняет степень результирующего отношения.

Расширенным декартовым произведением отношения R1 степени n со схемой $S_{R1} = (A_1, A_2, \dots, A_n)$ и отношения R2 степени m со схемой $S_{R2} = (B_1, B_2, \dots, B_m)$ называется отношение R3 степени n+m со схемой $S_{R3} = (A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$, содержащее кортежи, полученные сцеплением каждого кортежа q отношения R1 с каждым кортежем q отношения R2.

Операцию декартова произведения с учетом возможности перестановки атрибутов в отношении можно считать симметричной. Очень часто операция расширенного декартова самостоятельного значения не имеет. Результат выполнения данной операции обычно участвует в дальнейшей обработке.

В примере в отношении R7 задана обязательная номенклатура деталей для всех цехов, в отношении R8 дан перечень всех цехов. Тогда результирующее отношение R9 соответствует ситуации, когда каждый цех изготавливает все требуемые детали.

R7	
Шифр детали	Название детали
003412	Гайка M1
003415	Гайка M2

R8
Цех
Цех 1
Цех 2

R9		
Шифр детали	Название детали	Цех
003412	Гайка M1	Цех 1
003415	Гайка M2	Цех 1
003412	Гайка M1	Цех 2
003415	Гайка M2	Цех 2

3.2.3. Специальные операции реляционной алгебры

Первой специальной операцией реляционной алгебры является селекция (выбор, горизонтальный выбор, операция фильтрации, операция ограничения отношений).

Селекция (выбор) – это операция создания нового отношения, включающего те кортежи исходного отношения, для которых истинно условие выбора или фильтрации.

Например, выбрать из R9 детали с шифром 003412.

$R_{10} = R_9[\text{Шифр детали} = \text{«003412»}]$

R10		
Шифр детали	Название детали	Цех
003412	Гайка M1	Цех 1
003412	Гайка M1	Цех 2

Следующей специальной операцией является проекция.

Пусть R – отношение, $S_R = (A_1, A_2, \dots, A_n)$ – схема отношения R.

Обозначим через V подмножество $[A_i]$; $V \subseteq \{A_i\}$.

При этом пусть V^1 – множество атрибутов из $\{A_i\}$, не вошедших в V.

Проекцией отношения R на набор атрибутов V, обозначаемой $R[V]$, называется отношение со схемой, соответствующей набору атрибутов V $S_{R[V]} = V$, содержащему кортежи, получаемые из кортежей исходного отношения R путем удаления из них значений, не принадлежащих атрибутам из набора V.

По определению отношений все дублирующие кортежи удаляются из результирующего отношения.

Проекцию называют иногда операцией вертикального выбора, позволяющей получить только требуемые характеристики моделируемого объекта. Чаще всего проекция используется как промежуточный шаг в операциях горизонтального выбора, или фильтрации. Кроме того, она используется самостоятельно на заключительном этапе получения ответа на запрос.

Например, выберем все цеха, которые изготавливают деталь «Гайка М1». Воспользуемся отношением R10, полученным при реализации операции выбора, и выполним проекцию на столбец «Цех». Результатом выполнения этих операций будет отношение R11: $R11 = R10[\text{Цех}]$.

R11
Цех
Цех 1
Цех 2

Следующей специальной операцией реляционной алгебры является операция соединения.

В отличие от рассмотренных специальных операций реляционной алгебры: выбора и проекции, которые являются унарными, то есть производятся над одним отношением, операция соединения является бинарной, то есть исходными для нее являются два отношения, а результатом – одно.

Пусть $R = \{r\}$, $Q = \{q\}$ – исходные отношения, S_R , S_Q – схемы отношений R и Q соответственно.

$S_R = (A_1, A_2, \dots, A_k)$; $S_Q = (B_1, B_2, \dots, B_m)$,

где A_i , B_j – имена атрибутов в схеме отношений R и Q соответственно.

При этом полагаем, что заданы наборы атрибутов A и B $A \subseteq \{A_i\}_{i=1,k}$; $B \subseteq \{B_j\}_{j=1,m}$, и эти наборы состоят из Θ -сравнимых атрибутов.

Тогда **соединением** отношений R и Q при условии β будет подмножество декартова произведения отношений R и Q, кортежи которого удовлетворяют условию β . Соединение обозначают следующим образом: $R [\beta] Q$.

Например, рассмотрим следующий запрос. Пусть отношение R12 содержит перечень деталей с указанием материалов, из которых эти детали изготавливаются.

R12		
Шифр детали	Название детали	Материал
003412	Гайка M1	Сталь-ст1
003415	Гайка M2	Сталь-ст2
003416	Гайка M3	Сталь-ст1
003477	Болт M1	Сталь-ст2

Получить перечень деталей, которые изготавливаются в цехе 1 из материала «сталь-ст1»:

$R13 = (R12[(R12.Шифр\ детали = R9.Шифр\ детали) \cap R9.Цех = \text{“Цех 1”} \cap R12.Материал = \text{«сталь-ст1»}]R9)[Название\ детали]$.

В результате получим Гайка M1.

Важными с практической точки зрения частными случаями соединения являются эквисоединение и естественное соединение.

Операция эквисоединения характеризуется тем, что условие сравнения задает равенство операндов.

Операция естественного соединения применяется к двум отношениям, имеющим общий атрибут (простой или составной). Этот атрибут в отношениях имеет одно и то же имя (совокупность имен) и определен на одном и том же домене (доменах).

Последней специальной операцией реляционной алгебры является деление.

Результат **деления** отношения R с атрибутами A и B на отношение Q с атрибутом B, где A и B простые или составные атрибуты, причем атрибут B – общий атрибут, определенный на одном и том же домене (множестве доменов составного атрибута), является отношение P с атрибутом A, состоящее из кортежей r таких, что в отношении R имеются кортежи (r,s), причем множество значений s включает множество значений атрибута B отношения Q.

Деление обозначают следующим образом: $P = R[A:B]Q$.

Например, пусть отношение R содержит сведения о деталях, выпускаемых в каждом цехе, отношение Q – сведения о номенклатуре всех выпускаемых деталей. Требуется определить перечень цехов, в которых выпускается вся номенклатура деталей. Тогда решением этой задачи будет операция деления отношения R на отношение Q по набору атрибутов (Шифр детали, Наименование детали).

$R = R [\text{Шифр детали, Наименование детали} : \text{Шифр детали, Наименование детали}]Q.$

R		
Шифр детали	Наименование детали	Цех
003412	Гайка М1	Цех 1
003415	Гайка М2	Цех 1
003416	Гайка М3	Цех 1
003477	Болт М1	Цех 1
003412	Гайка М1	Цех 2
003477	Болт М1	Цех 2

Q	
Шифр детали	Наименование детали
003412	Гайка М1
003415	Гайка М2
003416	Гайка М3
003477	Болт М1

P
Цех
Цех 1

Дополнительные операции реляционной алгебры, предложенные Дейтом, включают: переименование, расширение, подведение итогов, присвоение, вставку, обновление, удаление, реляционное сравнение.

3.3. Реляционное исчисление

Принципиальное различие между реляционной алгеброй и реляционным исчислением состоит в том, что в первом случае процесс получения искомого результата описывается явным образом путем указания набора операций, которые надо выполнить для получения результата, а во втором – указываются свойства искомого отношения без конкретизации процедуры его получения.

Преимуществом реляционного исчисления перед реляционной алгеброй можно считать то, что пользователю не требуется самому строить алгоритм выполнения запроса. Программа СУБД сама строит эффективный алгоритм.

Математической основой реляционного исчисления является исчисление предикатов – один из разделов математической логики. Понятие реляционного исчисления как языка работы с базами данных впервые предложено Коддом. Им же был разработан язык ALPHA – прототип программно реализованного языка QUEL, который некоторое время конкурировал с языком SQL.

Существует два варианта исчислений: *исчисление кортежей* и *исчисление доменов*. В первом случае для описания отношений используются переменные, допустимыми значениями которых являются кортежи отношения, а во втором случае – элементы домена.

Реляционное исчисление, основанное на кортежах (**исчисление кортежей**), предложено и реализовано при разработке языка ALPHA. В нем, как и в процедурных языках программирования, сначала нужно описать используемые переменные, а затем записывать выражения.

Описательную часть исчисления можно представить в виде

RANGE OF <переменная> IS <список>,

где прописными буквами записаны ключевые слова языка, <переменная > - идентификатор переменной кортежа (области зна-

чений), а <список> - последовательность одного или более элементов, разделенных запятыми, т.е. конструкции вида $X_1[, X_2[... , X_n]...]$.

Конструкция RANGE указывает идентификатор переменной и область ее допустимых значений. Список элементов $X_1[, X_2[... , X_n]...]$ содержит элементы, каждый из которых является либо отношением, либо выражением над отношением. Все элементы списка должны быть совместимы по типу, т.е. соответствующие элементам отношения должны иметь идентичные заголовки. Область допустимых значений <переменной> образуется путем объединения значений всех элементов списка. Так, запись вида RANGE OF T IS X_1, X_2 означает, что область определения переменной T включает в себя все значения из отношения, которое является объединением отношений X_1 и X_2 .

Пример 1. Варианты описаний.

RANGE OF D IS R9;

RANGE OF DC IS R10;

RANGE OF SD IS (R12) WHERE R12.Материал = 'Сталь-ст1';

Переменные D, DC в первых двух примерах определены соответственно на отношениях R9 и R10. Переменная SD может принимать значения из множества кортежей отношения R12, где материал совпадает с материалом Сталь-ст1.

Запись выражения, формирующего запрос на языке исчисления кортежей с помощью формы Бэкуса-Наура, упрощенно можно представить следующим образом:

<выражение> ::= ($y_1[, y_2[... , y_m]...]$) [WHERE wff]

y_i ::= {<переменная> | <переменная>, <атрибут> } [AS

<атрибут>] wff ::= <условие> | NOT wff |

<условие> AND wff | <условие> OR wff |

IF <условие> THEN wff | EXISTS <переменная> (wff) |

FORALL <переменная> (wff) | (wff)

Общий смысл записи выражения состоит в перечислении атрибутов результирующего (целевого) отношения, атрибуты которого должны удовлетворять условию истинности формулы

wff (well formulated formula – правильно построенная формула). Список атрибутов целевого отношения, или целевой список, в терминах реляционной алгебры по существу определяет операцию проекции, а формула wff – селекцию кортежей.

В паре <переменная>, <атрибут> первая составляющая служит для указания переменной кортежа (определенной конструкцией RANGE), а вторая – для определения атрибута отношения, на котором изменяется переменная кортежа. Необязательная часть AS <атрибут> используется для переименования целевого отношения. Если она отсутствует, то имя атрибута целевого отношения наследуется от соответствующего имени атрибута исходного отношения.

Употребление в качестве элемента целевого отношения T равносильно перечислению в списке всех соответствующих атрибутов, т.е. $T.A_1, T.A_2, \dots, T.A_n$, где $A_1, A_2, \dots, A_n$ – атрибуты отношения, сопоставляемого с переменной T.

Пример 2. Варианты записи пары <переменная>.<атрибут>.

SD.[Шифр детали]

DC.[Цех] AS Цех_изготовитель

В приведенном определении wff <условие> представляет собой либо формулу wff, заключенную в скобки, либо простое сравнение вида

<операнд1> Θ <операнд2>,

где в качестве любого операнда выступает переменная или скалярная константа, а символ Θ обозначает операцию сравнения $=, \neq, >, \geq, <, \leq$ и т.д.

Ключевые слова NOT, AND, OR обозначают логические операции соответственно: Не, И, ИЛИ. Ключевые слова IF и THEN позволяют задать условие. Ключевые слова EXISTS и FORALL называются **кванторами**. Первый из них – квантор существования, а второй – квантор всеобщности.

Формула wff вида: EXISTS x (f) означает: «Существует по крайней мере одно такое значение переменной x, что вычисление формулы f дает значение истина».

Выражение вида: $\text{FORALL } x(f)$ интерпретируется как высказывание: «Для всех значений переменной x вычисление формулы f дает значение истина».

В общем случае переменные кортежей в формулах могут быть свободными или связанными. В формулах $\text{EXISTS } x(f)$ и $\text{FORALL } x(f)$ переменные кортежей x всегда являются связанными.

Пример 3. Запись выражения.

Приведем запись выражения, соответствующего запросу: «Получить названия цехов, где производится деталь «Гайка М1».

$D.[\text{Цех}] \text{ WHERE EXISTS } R9 (D.[\text{Название детали}] = \text{'Гайка М1'})$

Вариант реляционного исчисления, основанного на доменах (**исчисление доменов**), предложен Лакроиксом и Пиротте (Lacroix and Pirotte), которые также разработали на его основе соответствующий язык ILL. Другими языками, основанными на исчислении доменов, являются: FQL, DEDUCE, QBE. По утверждению Дейта, язык QBE включает элементы исчисления кортежей и исчисления доменов, но более близок ко второму. Он не является реляционно полным, так как не поддерживает операцию отрицания квантора существования (NOT EXISTS).

Исчисление доменов имеет много сходства с исчислением кортежей. В отличие от исчисления кортежей, в исчислении доменов основой любого выражения запроса выступают переменные доменов. Переменная домена – это скалярная переменная, значения которой охватывают элементы некоторого домена.

Большая часть различий рассматриваемых исчислений заключается в том, что исчисление доменов поддерживает дополнительную форму условия, называемую *условием принадлежности*. В общем виде условие принадлежности записывается следующим образом:

$$R(A_1:v_1, A_2:v_2, \dots),$$

где A_i – атрибут отношения R , а v_i – переменная домена или литерал. Проверяемое условие истинно, если и только если существует кортеж в отношении R , имеющий атрибуты A , равные заданным в выражении соответствующим значениям v .

Например, выражение $R12$ ([Шифр детали] : '003412', Материал : 'Сталь-ст1') истинно, если в отношении $R12$ существует хотя бы один кортеж со значением '003412' атрибута Шифр детали и значением 'Сталь-ст1' атрибута Материал. Аналогично, выражение $R12$ ([Шифр детали] : SHD, Материал : M) истинно, если в отношении $R12$ существует кортеж, в котором значение атрибута [Шифр детали] эквивалентно текущему значению переменной домена SHD, а значение атрибута Материал эквивалентно текущему значению переменной домена M.

В следующих примерах будем подразумевать существование (объявленное каким-либо образом, подобно оператору RANGE исчисления кортежей) следующих переменных доменов: SHD (домен атрибута Шифр детали), HD (домен атрибута Название детали), M (домен атрибута Материал).

Пример 4. Выражения исчисления доменов.

(SHD) WHERE $R12$ ([Шифр детали]: SHD)

(SHD) WHERE $R12$ ([Шифр детали] : SHD, Материал : 'Сталь-ст1')

Первое выражение означает множество всех шифров деталей отношения $R12$, второе – множество шифров деталей, изготовленных из материала "Сталь-ст1".

3.4. Язык запросов по образцу QBE

Хранимые в базе данные можно обрабатывать вручную, последовательно просматривая и редактируя данные в таблицах с помощью имеющихся в СУБД соответствующих средств. Для повышения эффективности обработки данных применяют запросы, позволяющие производить множественную обработку данных. Запросы позволяют одновременно вводить, редактиро-

вать и удалять множество записей, а также выбирать данные из таблиц.

Запрос представляет собой специальным образом описанное требование, определяющее состав производимых над базой данных операций по выборке, удалению или модификации хранимых данных.

Для подготовки запросов с помощью различных СУБД чаще всего используются два основных языка описания запросов:

- язык QBE (Query By Example) – язык запросов по образцу;
- SQL (Structured Query Language) – структурированный язык запросов.

По возможностям манипулирования данными при описании запросов указанные языки практически эквивалентны. Главное отличие между ними заключается в способе формирования запросов: язык QBE предполагает ручное или визуальное формирование запроса, в то время как использование SQL означает программирование запроса.

Характеристика языка QBE. Теоретической основой языка QBE является реляционное исчисление с переменными доменами. Язык QBE позволяет задавать сложные запросы к базе данных путем заполнения предлагаемой СУБД запросной формы. Такой способ задания запросов обеспечивает высокую наглядность и не требует указания алгоритма выполнения операции – достаточно описать образец желаемого результата. В каждой из современных реляционных СУБД имеется свой вариант языка QBE. На языке QBE можно задавать однотабличные и многотабличные запросы.

С помощью запросов на языке QBE можно выполнять следующие основные операции:

- выборку данных;
- вычисления над данными;
- вставку новых записей;
- удаление записей;

- модификацию (изменение) данных.

Результатом выполнения запроса является новая таблица, называемая ответной (первые две операции), или обновленная исходная таблица (остальные операции).

Выборка, вставка, удаление и модификация могут производиться безусловно или в соответствии с условиями, задаваемыми с помощью логических выражений. Вычисления над данными задаются с помощью арифметических выражений и порождают в ответных таблицах новые поля, называемые вычисляемыми.

В современных СУБД, например, в Access и Visual FoxPro, многие действия по подготовке запросов с помощью языка QBE выполняются визуально с помощью мыши.

Перспективы развития языка QBE. Анализ современных СУБД позволяет предположить следующие направления развития языка QBE:

- повышение наглядности и удобства;
- появление средств, соответствующих новым возможностям СУБД, например, формулировка неточных или нечетких запросов, манипулирование большими объемами данных;
- использование новых типов данных (графических, аудио-, видео- и др.);
- применение в ближайшем будущем ограниченного естественного языка формулировки запросов;
- в более отдаленной перспективе использование речевого ввода запросов.

3.5. Структурированный язык запросов SQL

3.5.1. История развития SQL

История развития SQL. Язык SQL – стандартный язык запросов по работе с реляционными базами данных. Язык SQL появился после реляционной алгебры, и его прототип был разработан в конце 70-х годов в компании IBM Research. Он был реализован в первом прототипе реляционной СУБД фирмы IBM

System R. В дальнейшем этот язык применялся во многих коммерческих СУБД и в силу своего широкого распространения постепенно стал стандартом «де-факто» для языков манипулирования данными в реляционной СУБД.

Первый международный стандарт языка SQL был принят в 1989 г. (SQL89 или SQL1).

В конце 1992 г. был принят новый международный стандарт языка SQL (SQL92 или SQL2). В настоящее время большинство производителей СУБД внесли изменения в свои продукты так, чтобы они в большей степени удовлетворяли стандарту SQL2.

В 1999 году появился новый стандарт, названный SQL3. Стандарт SQL3 соответствует качественным серьезным преобразованиям. В SQL3 введены новые типы данных, при этом предполагается возможность задания сложных структурированных типов данных, которые в большой степени соответствуют объектной ориентации. Наконец, добавлен раздел, который вводит стандарты на события и триггеры. В стандарте определены возможности четкой спецификации триггеров как совокупности события и действия. В качестве действия могут выступать не только последовательность операторов SQL, но и операторы управления ходом выполнения программы. В рамках управления транзакциями произошел возврат к старой модели транзакций, допускающей точки сохранения, и возможность указания в операторе отката ROLLBACK точек возврата не только в начало транзакции, но и в промежуточную ранее сохраненную точку.

3.5.2. Общая характеристика языка

Язык SQL предназначен для выполнения операций над таблицами (создание, удаление, изменение структуры) и над данными таблиц (выборка, изменение, добавление и удаление), а также некоторых сопутствующих операций. SQL является процедурным языком и не содержит операторов управления,

организации подпрограмм, ввода-вывода и т.п. В связи с этим SQL автономно не используется, обычно он погружен в среду встроенного языка программирования СУБД (например, FoxPro СУБД Visual FoxPro, ObjectPAL СУБД Paradox, Visual Basic for Application СУБД Access).

В современных СУБД с интерактивным интерфейсом можно создавать запросы, используя другие средства, например QBE. Однако применение SQL зачастую позволяет повысить эффективность обработки данных в базе. Например, при подготовке запроса в среде Access можно перейти из окна Конструктора запросов (формулировки запроса по образцу на языке QBE) в окно с эквивалентным оператором SQL.

Язык SQL не обладает функциями полноценного языка разработки, а ориентирован на доступ к данным, поэтому его включают в состав средств разработки программ. В этом случае его называют встроенным SQL. Стандарт языка SQL поддерживают современные реализации следующих языков программирования: PL/1, Ada, C, COBOL, Fortran, Pascal.

В специализированных системах разработки приложений типа клиент-сервер среда программирования, кроме того, обычно дополнена коммуникационными средствами (установление и разъединение соединений с серверами БД, обнаружение и обработка возникающих в сети ошибок и т.д.), средствами разработки пользовательских интерфейсов, средствами проектирования и отладки.

Различаются два основных метода использования встроенного SQL: статистический и динамический.

При статическом использовании языка (статический SQL) в тексте программы имеются вызовы функций языка SQL, которые жестко включаются в выполняемый модуль после компиляции.

При динамическом использовании языка (динамический SQL) предполагается динамическое построение вызовов SQL-функций и интерпретация этих вызовов, например, обращение к данным удаленной базы, в ходе выполнения программы.

Основным назначением языка SQL (как и других языков для работы с базами данных) является подготовка и выполнение запросов. В результате выборки данных из одной или нескольких таблиц может быть получено множество записей, называемое представлением.

Представление, по существу является таблицей, формируемой в результате выполнения запроса. Оно является разновидностью хранимого запроса. По одним и тем же таблицам можно построить несколько представлений. Само представление описывается путем указания идентификатора представления и запроса, который должен быть выполнен для его получения.

Для удобства работы с представлениями в язык SQL введено понятие курсора. **Курсор** представляет собой своеобразный указатель, используемый для перемещения по наборам записей при их обработке. Описание и использование курсора в языке SQL выполняется следующим образом. В описательной части программы выполняют связывание переменной типа курсор (CURSOR) с оператором SQL (обычно с оператором SELECT). В выполняемой части программы производится открытие курсора (OPEN <имя курсора>), перемещение курсора по записям (FETCH <имя курсора>...), сопровождаемое соответствующей обработкой, и, наконец, закрытие курсора (CLOSE <имя курсора>).

3.5.3. Структура SQL

В отличие от реляционной алгебры, где были представлены только операции запросов к БД, SQL является полным языком. Операторы языка SQL можно условно разделить на два подязыка: язык определения данных (Data Definition Language – DDL) и язык манипулирования данными (Data Manipulation Language – DML). Основные операторы языка SQL представлены в табл.3.1. Кроме того, язык содержит операторы управле-

ния транзакциями, администрирования данных и управления курсором.

Таблица 3.1

Операторы языка SQL

Вид	Оператор	Назначение	Действие
DDL	CREATE TABLE	Создание таблицы	Создание новой таблицы в БД
	DROP TABLE	Удаление таблицы	Удаление таблицы из БД
	ALTER TABLE	Изменение структуры таблицы	Изменение структуры существующей таблицы или ограничений целостности, задаваемые для данной таблицы
	CREATE VIEW	Создание представления	Создание виртуальной таблицы, соответствующей некоторому SQL-запросу
	DROP VIEW	Удаление представления	Удаление ранее созданного представления
	ALTER VIEW	Изменение представления	Изменение ранее созданного представления
	CREATE INDEX	Создание индекса	Создание индекса для некоторой таблицы для обеспечения быстрого доступа по атрибутам, входящим в индекс
	DROP INDEX	Удаление индекса	Удаление ранее созданного индекса

Продолжение табл. 3.1

Вид	Оператор	Назначение	Действие
DML	SELECT	Выборка записей	Формирование результирующего отношения, соответствующего запросу (оператор, заменяющий все операторы реляционной алгебры)
	UPDATE	Изменение записей	Обновление значения одного или нескольких столбцов в одной или нескольких строках, соответствующих условиям фильтрации
	INSERT	Вставка новых записей	Вставка одной строки в базовую таблицу. Допустимы модификации оператора, при которых сразу несколько строк могут быть перенесены из одной таблицы или запроса в базовую таблицу
	DELETE	Удаление записей	Удаление одной или нескольких строк, соответствующих условиям фильтрации, из базовой таблицы. Применение оператора согласуется с принципами поддержки целостности, поэтому этот оператор не всегда может быть выполнен корректно, даже если синтаксически он записан правильно

3.5.4. Оператор выбора SELECT

Язык запросов в SQL состоит из единственного оператора SELECT.

Синтаксис оператора SELECT имеет следующий вид:

```
SELECT [ ALL| DISTINCT] <Список полей>|*  
FROM <Список таблиц>  
[WHERE <Предикат-условие выборки или соедине-  
ния>]  
[GROUP BY <Список полей результата>]  
[HAVING <Предикат-условие для группы>]  
[ORDER BY <Список полей, по которым  
упорядочить вывод>];
```

SELECT – ключевое слово, которое сообщает СУБД, что эта команда – запрос. Все запросы начинаются с этого слова с последующим пробелом. За ним может следовать способ выборки.

Здесь ключевое слово **ALL** означает, что в результирующий набор строк включаются все строки, удовлетворяющие условиям запроса. Значит, в результирующий набор могут попасть одинаковые строки. Это нарушение принципов теории отношений (в отличие от реляционной алгебры, где по умолчанию предполагается отсутствие дубликатов в каждом результирующем отношении). Ключевое слово **ALL** действует по умолчанию, значит, его можно не писать. Таким образом, если **ALL** отсутствует, то выбираются все строки, удовлетворяющие условиям отбора.

Ключевое слово **DISTINCT** означает, что в результирующий набор включаются только различные строки, то есть дубликаты строк результата не включаются в набор.

Список полей – это список перечисленных через запятую столбцов, которые выбираются запросом из таблиц.

Символ * (звездочка) означает, что в результирующий набор включаются все столбцы из исходных таблиц запроса. Часто требуется перед символом «*» указывать имя таблицы и символ «.», а затем символ «*».

В разделе **FROM** задается перечень исходных отношений (таблиц) запроса. В случае если указано более одного имени таблицы, неявно подразумевается, что над перечисленными таблицами осуществляется операция декартова произведения.

Разделы SELECT и FROM являются обязательными, все другие разделы являются необязательными.

Примеры.

Выражение

```
SELECT Студенты.* FROM Студенты ;
```

означает выбор всех полей из таблицы Студенты.

Выражение

```
SELECT NOM_ZACH, FIO FROM Студенты ;
```

означает выбор двух полей из таблицы Студенты.

Выражение

```
SELECT Студенты.*, Экзамены.*
```

```
FROM Студенты, Экзамены;
```

соответствует декартову произведению таблиц Студенты и Экзамены; при этом выбираются все поля из таблицы Студенты и все поля из таблицы Экзамены. В данном случае к каждой записи из таблицы Студенты будут приписаны все записи из таблицы Экзамены. Это является неправильным: к каждому студенту должны быть приписаны только его сдачи экзаменов. Для этого надо писать условие соединения двух таблиц (указывать поля, по которым таблицы связаны между собой).

```
SELECT Студенты.*, Экзамены.*
```

```
FROM Студенты, Экзамены
```

```
WHERE Студенты.Номер_зачетки = Экзамены.Номер_зачетки
```

Часто в разделе FROM указывают второе (алиасное) более короткое имя таблицы и его используют перед именем поля или символом «*». Например:

```
SELECT С.*, Э.*  
FROM Студенты С, Экзамены Э  
WHERE С.Номер_зачетки = Э.Номер_зачетки
```

В разделе **WHERE** задаются условия отбора строк результата или условия соединения кортежей исходных таблиц, подобно операции условного соединения в реляционной алгебре.

В выражении условий раздела **WHERE** могут быть использованы следующие предикаты.

Предикаты сравнения (=, <, >, >=, <=), которые имеют традиционный смысл.

Пример 1. Выбрать из таблицы Продажа все поля для записей, где поле Количество больше 10:

```
Select Продажа.* from Продажа where Количество > 10;
```

Пример 2. Выбрать из таблицы Экзамен все поля для записей, где оценка 5:

```
Select Экзамен.* from Экзамен where Оценка = 5;
```

Предикат Between A and B – принимает значения между A и B. Предикат истинен, когда сравниваемое значение попадает в заданный диапазон, включая границы диапазона. Одновременно в стандарте задан и противоположный предикат **Not Between A and B**, который истинен тогда, когда сравниваемое значение не попадает в заданный интервал, включая его границы.

Пример 1. Выбрать из таблицы Продажа все поля для записей, где поле Количество попадает в интервал от 10 до 50:

```
Select Продажа.* from Продажа  
where Количество between 10 and 50;
```

Пример 2. Выбрать из таблицы Продажа все поля для записей, где поле Дата продажи попадает в интервал от 1.01.06 до 31.01.06:

```
Select Продажа.* from Продажа  
where [Дата продажи] between #01/01/06# and #31/01/06#;
```

Предикат вхождения в множество IN (множество) истинен тогда, когда сравниваемое значение входит в множество заданных значений. При этом множество значений может быть задано простым перечислением или встроенным подзапросом. Одновременно существует противоположный предикат NOT IN (множество), который истинен тогда, когда сравниваемое значение не входит в заданное множество.

Пример 1. Выбрать из таблицы Группы все поля для записей, где поле Шифр группы имеет значения ВМ-051, ВМ-052, ВМ-053:

Select Группы.* from Группы where [Шифр группы] in (“ВМ-051”, “ВМ-052”, “ВМ-053”);

Пример 2. Выбрать из таблицы Экзамен все поля для записей, где поле Оценка принимает значения 4 или 5:

Select Экзамен.* from Экзамен where Оценка in (4, 5);

Предикаты сравнения с образцом LIKE и NOT LIKE. Предикат LIKE требует задания шаблона, с которым сравнивается заданное значение, предикат истинен, если сравниваемое значение соответствует шаблону, и ложен в противном случае. Предикат NOT LIKE имеет противоположный смысл. Шаблон может содержать % (* для Access) для обозначения любого числа любых символов; (?) для Access) для обозначения любого одного символа.

Пример. Выбрать из таблицы Студенты все поля для записей, где поле Фамилия начинается с буквы «С» или «М»:

Для СУБД Access

Select Студенты.* from Студенты

Where Фамилия = like “С*” or Фамилия = like “М*”;

Для других СУБД

Select Студенты.* from Студенты

Where Фамилия = like “С%” or Фамилия = like “М%”;

Предикат сравнения с неопределенным значением IS NULL. Для выявления равенства значения некоторого атрибута

неопределенному значению применяют специальные стандартные предикаты:

<имя атрибута> IS NULL и <имя атрибута> IS NOT NULL

Пример. Выбрать из таблицы Сотрудники все поля для записей, где поле Домашний телефон не пусто:

**Select Сотрудники.* from Сотрудники
where [Домашний телефон] is not null;**

Предикат существования EXIST и не существования NOT EXIST.

В условиях поиска могут быть использованы все рассмотренные предикаты.

Рассмотрим базу данных, которая моделирует сдачу сессии в некотором учебном заведении. Пусть она состоит из трех отношений R1, R2, R3, представленных таблицами R1, R2, R3 соответственно.

R1 = (ФИО, Дисциплина, Оценка);

R2 = (ФИО, Группа);

R3 = (Группа, Дисциплина).

R1		
ФИО	Дисциплина	Оценка
Петров Ф.И.	Базы данных	5
Сидоров К.А.	Базы данных	4
Миронов А.В.	Базы данных	2
Петров Ф.И.	Моделирование	5
Сидоров К.А.	Моделирование	4
Миронов А.В.	Моделирование	Null
Трофимов П.А.	Сети ЭВМ	4
Иванова Е.А.	Сети ЭВМ	5
Уткина Н.В.	Сети ЭВМ	5

R2	
ФИО	Группа
Петров Ф.И.	ВМ-021
Сидоров К.А.	ВМ-021
Миронов А.В.	ВМ-021
Трофимов П.А.	ВМ-022
Иванова Е.А.	ВМ-022
Уткина Н.В.	ВМ-022

R3	
Группа	Дисциплина
ВМ-021	Базы данных
ВМ-021	Моделирование
ВМ-022	Сети ЭВМ

Приведем несколько примеров использования оператора SELECT.

Вывести список всех групп (без повторений), где должны пройти экзамены.

SELECT DISTINCT Группа FROM R3;

Результат:

Группа
ВМ-021
ВМ-022

Вывести список студентов, которые сдали экзамен по дисциплине «Базы данных» на «отлично».

SELECT ФИО FROM R1

WHERE Дисциплина = «Базы данных» AND Оценка = 5;

Результат:

ФИО
Петров Ф.И.

Найти студентов, пришедших на экзамен, но не сдавших его с указанием названия дисциплины. Оператор SELECT будет выглядеть следующим образом:

```
SELECT ФИО, Дисциплина
FROM R1
WHERE Оценка IS NULL;
Результат:
```

Миронов А.В.	Моделирование
--------------	---------------

Вывести список всех студентов, которым надо сдавать экзамены с указанием названий дисциплин, по которым должны проводиться эти экзамены.

```
SELECT ФИО, Дисциплина
FROM R2, R3
WHERE R2.Группа = R3.Группа;
```

Здесь WHERE задает условие соединения отношений R2 и R3, при отсутствии условий соединения результат будет эквивалентен декартову произведению, и в этом случае каждому студенту были бы приписаны все дисциплины из отношения R3, а не те, которые должна сдавать группа, в которой он учится.

Результат:

ФИО	Дисциплина
Петров Ф.И.	Базы данных
Сидоров К.А.	Базы данных
Миронов А.В.	Базы данных
Петров Ф.И.	Моделирование
Сидоров К.А.	Моделирование
Миронов А.В.	Моделирование
Трофимов П.А.	Сети ЭВМ
Иванова Е.А.	Сети ЭВМ
Уткина Н.В.	Сети ЭВМ

Оператор Select может содержать вычисляемые поля.
Для вычисляемого поля можно задать имя после слова AS.

Пример 1. Вывести все поля из таблицы Продажа и добавить вычисляемое поле Стоимость покупки:

Select П.*, [Цена за единицу] * [Количество] as [Стоимость покупки] from Продажа П;

Пример 2. Даны две таблицы Товары (Код, Название, Розничная цена), Продажа (Чек, Код, Дата продажи, Количество). Выбрать поля Название и Розничная цена из таблицы Товары и поля Чек, Дата продажи и Количество из таблицы Продажа. Добавить вычисляемое поле Стоимость покупки.

Select Т.Название, Т.[Розничная цена], П.Чек, П.[Дата продажи], П.Количество, Т.[Розничная цена] * П.Количество as [Стоимость покупки]

from Товары Т, Продажа П

where Т.Код = П.Код;

При выборе полей из разных таблиц необходимо:

указывать имя таблицы, затем ставить точку и указывать имя поля;

в where указывать условие соединения двух таблиц (в примере – это равенство полей Код из двух таблиц).

Пример 3. Даны две таблицы Knigi (Shifr, Avtor, Nazv, Cena) и Postavka (Nomer_posr, Shifr, Data_post, Kol). Выбрать все поля из двух таблиц и добавить вычисляемое поле Stoim_post (стоимость поставки).

Select К.*, Р.*, К.Cena * Р.Kol as Stoim_post

From Knigi К, Postavka Р

Where К.Shifr = Р.Shifr;

В данном примере К и Р – это алиасные (вторые) имена таблиц Knigi и Postavka соответственно.

3.5.5. Применение агрегатных функций и группировки

В разделе **GROUP BY** задается список полей группировки. **GROUP BY** группирует записи данных и объединяет в одну запись все записи данных, которые содержат идентичные значения в указанном поле (или полях).

В SQL добавлены дополнительные функции, которые позволяют вычислять обобщенные групповые значения. Для при-

менения агрегатных функций предполагается предварительная операция группировки. При группировке все множество кортежей отношения разбивается на группы, в которых объединяются кортежи, имеющие одинаковые значения атрибутов, которые заданы в списке группировки.

Например, сгруппируем отношение R1 по значению столбца Дисциплина. Мы получим 3 группы, для которых можем вычислить некоторые групповые значения, например количество кортежей в группе, максимальное или минимальное значение столбца Оценка.

Это осуществляется с помощью агрегатных функций. Агрегатные функции вычисляют одиночные значения для каждой группы таблицы. Список агрегатных функций представлен в табл. 3.2.

Таблица 3.2

Агрегатные функции

Функция	Результат
COUNT	Количество строк или непустых значений полей, которые выбрал запрос
SUM	Сумма всех выбранных значений данного поля
AVG	Среднеарифметическое значение всех выбранных значений данного поля
MIN	Наименьшее из всех выбранных значений данного поля
MAX	Наибольшее из всех выбранных значений данного поля

Агрегатные функции применяются подобно именам полей в операторе SELECT, но они используют имя поля как аргумент. С функциями SUM И AVG могут использоваться только числовые поля. С функциями COUNT, MAX, MIN могут использоваться как числовые, так и символьные поля. При использовании с символьными полями MAX и MIN будут транс-

лизовать их в эквивалент ASCII кода и обрабатывать в алфавитном порядке.

Например, можно вычислить количество студентов, сдавших экзамены по каждой дисциплине. Для этого надо выполнить запрос с группировкой по полю «Дисциплина» и вывести в качестве результата название дисциплины и количество строк в группе по данной дисциплине. Применение символа * в качестве аргумента функции COUNT означает подсчет всех строк в группе.

```
SELECT R1.Дисциплина, COUNT(*)
FROM R1
GROUP BY R1.Дисциплина;
```

Результат:

Дисциплина	COUNT(*)
Базы данных	3
Моделирование	3
Сети ЭВМ	3

Если же необходимо определить количество студентов, сдавших экзамен по каждой дисциплине, то необходимо исключить неопределенные значения из исходного отношения перед группировкой. В этом случае запрос будет выглядеть следующим образом:

```
SELECT R1.Дисциплина, COUNT(*)
FROM R1
WHERE R1.Оценка IS NOT NULL
GROUP BY R1.Дисциплина;
```

Результат:

Дисциплина	COUNT(*)
Базы данных	3
Моделирование	2
Сети ЭВМ	3

В этом случае строка со студентом

Миронов А.В.	Моделирование	Null
--------------	---------------	------

не попадет в набор кортежей перед группировкой, поэтому количество кортежей в группе для дисциплины «Моделирование» будет на 1 меньше.

Можно применять агрегатные функции также и без операции предварительной группировки, в этом случае все отношение рассматривается как одна группа и для этой группы можно вычислить одно значение на группу. Найдем количество студентов, успешно сдавших экзамены:

```
SELECT COUNT(*)  
FROM R1 WHERE Оценка > 2;
```

Аргументом агрегатных функций могут быть отдельные столбцы таблиц. Но для того, чтобы вычислить, например, количество различных значений некоторого столбца в группе, необходимо применить ключевое слово **DISTINCT** совместно с именем столбца. Вычислим количество различных оценок, полученных по каждой дисциплине:

```
SELECT R1.Дисциплина, COUNT (DISTINCT R1.Оценка)  
FROM R1 WHERE R1.Оценка IS NOT NULL  
GROUP BY R1.Дисциплина;
```

Результат:

Дисциплина	COUNT(DISTINCT R1.Оценка)
Базы данных	3
Моделирование	2
Сети ЭВМ	2

В результат можно включить значение поля группировки и несколько агрегатных функций, а в условиях группировки можно использовать несколько полей. При этом группы образуются по набору заданных полей группировки. Операции с агрегатными функциями могут быть применимы к нескольким исходным таблицам. Например, определить для каждой группы и каждой дисциплины количество студентов, успешно сдавших экзамен, и средний балл по дисциплине.

```

SELECT R2.Группа, R1. Дисциплина, COUNT(*),
AVG(Оценка) FROM R1, R2
WHERE R1.ФИО = R2.ФИО AND
R1.Оценка IS NOT NULL AND R1.Оценка >2
GROUP BY R2.Группа, R1.Дисциплина;

```

Результат:

Группа	Дисциплина	COUNT(*)	AVG (Оценка)
BM-021	Базы данных	2	3,67
BM-021	Моделирование	2	4,5
BM-022	Сети ЭВМ	3	4,67

Рассмотрим в качестве другого примера отношения-таблицы F и Q из базы данных «Банк», в которой содержится информация о счетах в филиалах некоторого банка:

F = (N, ФИО, Филиал, ДатаОткрытия, ДатаЗакрытия, Остаток);

Q = (Филиал, Город).

Определим суммарный остаток на счетах в филиалах:

```

SELECT Филиал, SUM(Остаток)
FROM F GROUP BY Филиал;

```

Группа состоит из строк с одинаковым значением поля Филиал, и функция SUM применяется отдельно для каждой такой группы, то есть суммарный остаток на счетах подсчитывается отдельно для каждого филиала. Значение поля, к которому применяется GROUP BY, имеет, по определению, только одно значение на группу вывода, как и результат работы агрегатной функции. Поэтому мы можем совместить в одном запросе агрегат и поле.

Рассмотрим еще несколько примеров на применение группировки и агрегатных функций.

Пусть даны следующие таблицы:

Дисциплины (Код_дисциплины, Название_дисциплины);
Группы (Шифр_группы, Факультет, Специальность,
Курс);

Студенты (Номер_зачетки, Шифр_группы, Фамилия. Имя,
Отчество, Дата_рождения);

Экзамены (Код_дисциплины, Номер_зачетки, Семестр,
Дата, Оценка).

Ключевые поля подчеркнуты. Таблицы Группы и Студенты связаны по полю Шифр_группы, таблицы Студенты и Экзамены связаны по полю Номер_зачетки, таблицы Дисциплины и Экзамены связаны по полю Код_дисциплины.

Пример 1. Определить количество студентов в каждой группе.

Select Шифр_группы, count(Номер_зачетки) As Кол_студентов

From Студенты Group by Шифр_группы;

Пример 2. Определить количество студентов в каждой группе и дополнительно вывести факультет.

**Select Факультет, С.Шифр_группы,
count(Номер_зачетки) As Кол_студентов**

From Группы Г, Студенты С

**Where Г.Шифр_группы = С.Шифр_группы
Group by Факультет, С.Шифр_группы;**

Пример 3. Определить количество групп каждой специальности и курса.

Select Специальность, Курс, count(Шифр_группы)

As Кол_групп

From Группы

Group by Специальность, Курс;

Пример 4. Определить среднюю оценку по каждой дисциплине в каждой группе.

```

Select      Название_дисциплины,      Шифр_группы,
avg(Оценка) As Средняя_оценка
From Студенты С, Экзамен Э, Дисциплины Д
Where (С.Номер_зачетки=Э.Номер_зачетки) and
(Э.Код_дисциплины=Д.Код_дисциплины)
Group by Название_дисциплины, Шифр_группы;

```

В разделе **HAVING** задаются предикаты-условия, накладываемые на каждую группу. **HAVING** используется для фильтрации записей, полученных в результате группировки. **WHERE** определяет, какие записи должны участвовать в группировании, т.е. фильтрует до группирования. **HAVING** определяет, какие из получившихся в результате группировки записей будут включены в результирующую выборку, т.е. фильтрует записи после группирования.

Имеются две таблицы, связанные по полю Филиал.

F (N, ФИО, Филиал, ДатаОткрытия, ДатаЗакрытия, Остаток)

Q (Филиал, Город)

Пример 1. Определить суммарные значения остатков на счетах, которые превышают 5000. Чтобы увидеть суммарные остатки свыше 5000, необходимо использовать предложение **HAVING**. Предложение **HAVING** определяет критерии, используемые, чтобы удалять определенные группы из вывода, точно так же, как предложение **WHERE** делает это для индивидуальных записей.

```

SELECT Филиал, SUM(Остаток)
FROM F
GROUP BY Филиал
HAVING SUM(Остаток) > 5000;

```

Аргументы в предложении **HAVING** подчиняются тем же самым правилам, что и в предложении **SELECT**, где используется **GROUP BY**. Они должны иметь одно значение на группу вывода.

Пример 2. Определить суммарные остатки на счетах филиалов в Воронеже, Липецке, Курске:

```
SELECT Филиал, SUM(Остаток)
```

```
FROM F, Q
```

```
WHERE F.Филиал = Q.Филиал
```

```
GROUP BY Филиал
```

```
HAVING Город IN («Воронеж», «Липецк», «Курск»);
```

Результатом выполнения раздела HAVING является сгруппированная таблица, содержащая только те группы строк, для которых результат вычисления условия поиска есть TRUE.

Пример 3. Пусть даны две таблицы Группы (Шифр_группы, Специальность, Факультет), Студенты (Номер_зачетки, Шифр_группы, Фамилия, Имя, Отчество). Определить группы, где количество студентов больше 20.

```
Select Шифр_группы, count(Номер_зачетки)
```

```
As Кол_студентов From Студенты
```

```
Group by Шифр_группы
```

```
Having count(Номер_зачетки) > 20;
```

Пример 4. Определить количество студентов в каждой группе специальности ВМ.

```
Select Специальность, С.Шифр_группы,
```

```
count(Номер_зачетки) As Кол_студентов
```

```
From Студенты С, Группы Г
```

```
Where С.Шифр_группы = Г.Шифр_группы
```

```
Group by Специальность, С.Шифр_группы
```

```
Having Специальность = "ВМ";
```

3.5.6. Раздел ORDER BY и ключевое слово TOP

Раздел определяет порядок сортировки записей, включенных в выборку:

...ORDER BY поле1 [ASC / DESC]
[, поле2 [ASC / DESC] [...]]

Особенности:

- раздел не является обязательным, однако он обязательно используется с предикатом TOP;
- по умолчанию сортировка идет по возрастанию, можно явно указать возрастающую сортировку, записав ASC. Ключ DESC задает сортировку по убыванию;
- порядок перечисления полей задает иерархию уровней сортировки;
- ORDER BY – последняя директива в запросе.

Примеры.

Отсортировать записи по полю ФИО в алфавитном порядке:

```
SELECT R1.*  
FROM R1  
ORDER BY ФИО;
```

Отсортировать записи в порядке убывания значений в поле Оценка:

```
SELECT R1.*  
FROM R1  
ORDER BY Оценка DESC;
```

Ключевое слово **TOP** используется для отображения некоторого количества начальных и конечных записей из результирующего набора. Для ограничения числа записей в результирующем наборе ключевое слово TOP в запросах сочетается с предложением, указывающим порядок сортировки. Причем ключевое слово TOP можно комбинировать как с числом, означающим количество записей, так и с числом, означающим процентное содержание отображаемых записей.

Например, выбрать из таблицы Студенты (Номер_зачетки, Шифр_группы, ФИО, Год_выпуска, Средний_балл) 5 лучших студентов выпуска 2012 года:

```
SELECT TOP 5 ФИО, Шифр_группы  
FROM Студенты  
WHERE Год_выпуска = 2012  
ORDER BY Средний_балл DESC;
```

Число, используемое в предикате TOP, должно быть целым без знака. Без директивы ORDER BY в выборку попали бы любые 5 студентов выпуска 2012 года. Предикат TOP не разделяет записи, имеющие одинаковые значения при упорядочивании. Это значит, если 5-й, 6-й, 7-й студенты имеют одинаковый средний балл, то в выборке будет не 5, а 7 записей.

Можно использовать ключевое слово **PERCENT** для того, чтобы включить в выборку определенный процент из верхней или нижней части диапазона, отсортированного в соответствии с директивой ORDER BY.

Например, выбрать десять процентов записей выпуска 2012 года из таблицы Студенты:

```
SELECT TOP 10 PERCENT ФИО, Шифр_группы  
FROM Студенты  
WHERE Год_выпуска = 2012  
ORDER BY Средний_балл DESC;
```

3.5.7. Вложенные запросы

С помощью SQL можно вкладывать запросы внутрь друг друга. Обычно внутренний запрос генерирует значение, которое проверяется в предикате внешнего запроса (в предложении WHERE или HAVING), определяющего, верно оно или нет. Совместно с подзапросом можно использовать предикат EXISTS, который возвращает истину, если вывод подзапроса не пуст.

В части FROM оператора SELECT допустимо применять синонимы (алиасные имена) к именам таблицы, если при фор-

мировании запроса требуется более чем один экземпляр некоторого отношения. Синонимы задаются с использованием ключевого слова AS, которое может быть вообще опущено. Поэтому часть FROM может выглядеть следующим образом:

FROM R1 AS A, R1 AS B

Или

FROM R1 A, R1 B

Оба выражения эквивалентны и рассматриваются как применения оператора SELECT к двум экземплярам таблицы R1.

Запрос, содержащий подзапрос, называется сложным. В процессе его выполнения сначала выполняется подзапрос, а затем основной запрос. При создании сложного запроса необходимо следовать следующим правилам:

- подзапросы должны заключаться в круглые скобки;
- предложение Order by может быть использовано только в основном запросе;
- подзапросы, возвращающие более одной записи, могут использоваться только с многозначными операторами (например, с оператором Exists. Exists имеет значение истина, если выборка не пуста).

Рассмотрим примеры вложенных запросов.

Пример 1. Даны две таблицы: Товары (Код товара, Наименование, Цена) и Продажи (Чек, Код товара, Дата, Продано). Из таблицы Товары требуется отобрать товары (указать поля Наименование и Цена), для которых поле Продано превышает 10.

```
SELECT Наименование, Цена FROM Товары  
WHERE EXISTS (SELECT [Код товара] FROM Продажи  
WHERE (Продано > 10) AND (Товары.[Код товара]=  
Продажи.[Код товара]));
```

Пример 2. Выбрать из таблицы Продажи (указать поля Чек и Продано) товары с наименованием «Шоколад Путешествие».

```
SELECT Чек, Продано FROM Продажи
WHERE [Код товара] = (SELECT [Код товара] FROM То-
вары WHERE Наименование = “Шоколад Путешествие”);
```

Поскольку в таблице Продажи не содержится наименование товара, то сначала с помощью подзапроса обращаются к таблице Товары и определяют код товара заданного наименования, а затем в основном запросе выбирают поля из таблицы Продажи, в которых код товара совпадает с кодом, полученным в результате выполнения подзапроса.

3.5.8. Внутренние и внешние объединения

Стандарт SQL2 расширил понятие условного объединения. В стандарте SQL1 при объединении отношений использовались только условия, задаваемые в части WHERE оператора SELECT, и в этом случае в результирующее отношение попадали только сцепленные по заданным условиям кортежи исходных отношений, для которых эти условия были определены и истинны. Однако в действительности часто необходимо объединять таблицы таким образом, чтобы в результат попали все строки из первой таблицы, а вместо тех строк второй таблицы, для которых не выполнено условие соединения, в результат попадали бы неопределенные значения. Или наоборот, включаются все строки из правой (второй) таблицы, а отсутствующие части строк из первой таблицы дополняются неопределенными значениями. Такие объединения были названы внешними в противоположность объединениям, определенным стандартом SQL1, которые стали называться внутренними.

Внутреннее объединение (INNER JOIN) возвращает записи из двух таблиц, если значение первичного ключа первой таблицы соответствует значению внешнего ключа второй таблицы, связанной с первой.

В общем случае синтаксис части FROM в стандарте SQL2 выглядит следующим образом:

FROM <список исходных таблиц>|
 <выражение естественного объединения>|
 <выражение объединения>|
 <выражение перекрестного объединения>|
 <выражение запроса на объединение>

где

<список исходных таблиц> -
 <имя таблицы1> [имя синонима таблицы1] [...]
 [, <имя таблицыN> [имя синонима таблицыN]];
 <выражение естественного объединения> -
 <имя таблицы1> NATURAL { INNER | FULL [OUTER]
 LEFT [OUTER] | RIGHT [OUTER]} JOIN <имя таблицы2>|
 <выражение объединения> -
 <имя таблицы1> { INNER | FULL [OUTER] | LEFT [OUTER]
 | RIGHT [OUTER]} JOIN { ON условие | [USING
 (список столбцов)]} <имя таблицы2>
 <выражение перекрестного объединения> -
 <имя таблицы1> CROSS JOIN <имя таблицы2>
 <выражение запроса на объединение> -
 <имя таблицы1> UNION JOIN <имя таблицы2>

В этих определениях **INNER** – означает внутреннее объединение, **LEFT** – левое объединение, то есть в результат входят все строки первой таблицы, а части результирующих кортежей, для которых не было соответствующих значений во второй таблице, дополняются значениями NULL (не определено). Ключевое слово **RIGHT** означает правое внешнее соединение, и в отличие от левого соединения в этом случае в результирующее отношение включаются все строки второй таблицы, а недостающие части из первой таблицы дополняются неопределенными значениями. Ключевое слово **FULL** определяет полное внешнее объединение: левое и правое. При полном внешнем объединении выполняются и правое и левое внешние объединения и в результирующее отношение включаются все строки из первой таблицы, дополненные неопределенными значениями.

ми, и все строки из второй таблицы, также дополненные неопределенными значениями. Ключевое слово OUTER означает внешнее объединение, но если заданы ключевые слова FULL, LEFT, RIGHT, то объединение всегда считается внешним.

Примеры внутренних и внешних объединений

Пусть даны три таблицы. Ключевые поля подчеркнуты.

Товары (Индекс, Наименование). Поставщики (Код, Фирма, Директор, Телефон). Поставка (Номер, Индекс, Код, Дата, Количество). Таблицы Товары и Поставка связаны по полю Индекс. Таблицы Поставщики и Поставка связаны по полю Код.

Примеры таблиц.

Товары	
32	Молоко «Поддубный» 2.5 %
33	Молоко «Поддубный» 3.5 %
34	Молоко «Вкуснотеево» 2.5 %
45	Творог «Поддубный» 5 %
48	Творог Вкуснотеево» 9 %

Поставщики			
231	ВМЗ № 1	Петров А.И.	2-88-45-23
235	ВМЗ № 2	Васильев С.Н.	2-90-90-90
301	ООО «Каменка-молоко»	Кудрин А.Н.	2-77-90-55

Поставка				
10	32	231	12.09.12	200
11	33	231	12.09.12	100
12	34	235	12.09.12	150
14	48	235	12.09.12	10

Пример внутреннего объединения. Выбрать поля Номер, Наименование, Фирма, Количество.

SELECT Номер, Наименование, Фирма, Количество
 FROM Товары Inner Join (Поставщики Inner Join Постав-
 ка On Поставщики.Код = Поставка.Код)
 On Товары.Индекс = Поставка.Индекс;
 Результат.

Номер	Наименование	Фирма	Количество
10	Молоко «Поддубный» 2.5 %	ВМЗ № 1	200
11	Молоко «Поддубный» 3.5 %	ВМЗ № 1	100
12	Молоко «Вкуснотеево» 2.5 %	ВМЗ № 2	150
14	Творог Вкуснотеево» 9 %	ВМЗ № 2	10

Пример левого объединения. Выбрать все поля из таблицы Товары и поля Код, Дата, Количество из таблицы Поставка. Результат будет содержать список всех товаров и соответствующие поставки, если они есть. Если поставок нет, то пустые поля.

SELECT Товары.Индекс, Товары.Наименование, Постав-
 ка.Код, Поставка.Дата, Поставка.Количество
 FROM Товары Left Join Поставка
 On Товары.Индекс = Поставка.Индекс;
 Результат.

Индекс	Наименование	Код	Дата	Количество
32	Молоко «Поддуб- ный» 2.5 %	231	12.09.12	200
33	Молоко «Поддуб- ный» 3.5 %	231	12.09.12	100
34	Молоко «Вкусно- теево» 2.5 %	235	12.09.12	150
45	Творог «Поддуб- ный» 5 %			
48	Творог Вкусно- теево» 9 %	235	12.09.12	10

Внешние объединения в Delphi записываются следующим образом:

Левое внешнее объединение, при котором в выборку включаются все записи таблицы, имя которой указано слева от оператора OUTER JOIN.

```
...  
FROM <табл. 1> LEFT OUTER JOIN <табл. 2>  
ON <условие соединения таблиц>
```

Правое внешнее объединение, при котором в выборку включаются все записи таблицы, имя которой указано справа от оператора OUTER JOIN.

```
...  
FROM <табл. 1> RIGHT OUTER JOIN <табл. 2>  
ON <условие соединения таблиц>
```

Полное внешнее объединение означает, что в выборку включаются все записи из правой и левой таблиц.

```
...  
FROM <табл. 1> FULL OUTER JOIN <табл. 2>  
ON <условие соединения таблиц>
```

Пример левого объединения. Выбрать из таблицы Товары наименование всех товаров, а из таблицы Продажи – суммарное количество проданных товаров:

```
SELECT Товары.Наименование, Sum(Продажи.Продано)  
AS [Всего продано]  
FROM Товары LEFT OUTER JOIN Продажи  
ON Товары.[Код товара] = Продажи.[Код товара]  
GROUP BY Товары.Наименование;
```

Так как таблица Товары указана слева от оператора LEFT JOIN, то результирующая выборка будет содержать полный список наименований товаров, включая те, что ни разу не были проданы.

3.5.9. Перекрестные запросы

Перекрестный запрос – способ группировки данных по двум измерениям, позволяющий отображать итоги в компактном результирующем наборе. В перекрестном запросе группировка выполняется по двум полям (группировка по строкам может быть больше, чем по одному полю), а итоговая функция применяется к полю, выводимому на пересечении строк и столбцов. Структура перекрестного запроса следующая:

- в конструкции TRANSFORM указывается поле и групповая функция, применяемая к нему. Данное поле выводится на пересечении строк и столбцов;
- в Select указывается поле, выводимое в заголовках строк;
- в From указываются имена таблиц, из которых выбираются данные;
- в конструкции GROUP BY указывается поле, по которому проводится группировка и которое выводится в качестве заголовков строк.
- в конструкции PIVOT указывается поле, значения которого выводятся в качестве заголовков столбцов.

Пример 1. Дана таблица Продажа_бензина (Чек, Марка, Дата_продажи, Оператор, Количество). Определить суммарную продажу каждой марки бензина за каждый день. Марки бензина вывести в столбцах, даты – в строках, суммарную продажу – на пересечении строк и столбцов.

```
TRANSFORM SUM(Количество) AS SUM_Количество  
SELECT Дата_продажи  
FROM Продажа_бензина  
GROUP BY Дата_продажи  
PIVOT Марка;
```

Пример 2. Даны таблицы Продажа_бензина (Чек, Марка, Дата_продажи, Номер_сотрудника, Количество, Цена) и Сотрудники (Номер_сотрудника, Фамилия, Имя, Отчество, Долж-

ность, Оклад). Определить суммарную стоимостную продажу бензина каждым сотрудником за каждый день. В строках вывести даты, в столбцах – фамилии, на пересечении строки и столбца – суммарную стоимостную продажу.

```
TRANSFORM SUM(Количество * Цена) AS SUM_Стоим
SELECT Дата_продажи
FROM Продажа_бензина INNER JOIN Сотрудники
ON Продажа_бензина.[Номер_сотрудника] = Сотрудни-
ки.[Номер сотрудника]
GROUP BY Дата_продажи
PIVOT Фамилия;
```

3.5.10. Операторы манипулирования данными

В операции манипулирования данными входят три действия: операция удаления записей – ей соответствует оператор DELETE, операция добавления или ввода новых записей – ей соответствует оператор INSERT и операция изменения (обновления полей в записях) – ей соответствует оператор UPDATE.

Все операторы манипулирования данными позволяют изменить данные только в одной таблице.

Оператор ввода данных **INSERT** имеет следующий синтаксис:

```
INSERT INTO имя_таблицы [(<список столбцов>)]  
VALUES (<список значений>)
```

Пример 1. Ввод одной записи в таблицу Экзамены.

```
INSERT INTO Экзамены (ФИО, Дисциплина, Оценка)  
VALUES («Попова», «БД», 3);
```

Пример 2. При вводе полной строки можно не задавать список имен столбцов.

```
INSERT INTO Экзамены  
VALUES («Попова», «Моделирование», 3);
```

Пример 3. Можно ввести неполный перечень значений, в этом случае список имен вводимых столбцов задают обязательно, например:

```
INSERT INTO Экзамены (ФИО, Дисциплина)
VALUES («Бурковский», «Сети ЭВМ»);
```

В этом примере столбцу Оценка будет присвоено значение NULL.

Оператор ввода данных позволяет ввести сразу множество строк, если они выбираются из другой таблицы. Тогда вместо VALUES ставят SELECT.

Пример 4. Добавить из таблицы ПОСТАВКА в таблицу АРХИВ_ПОСТАВОК записи за 10 сентября 2012 года:

```
INSERT INTO АРХИВ_ПОСТАВОК
(Индекс_товара,Код_поставщика,Дата_поступления, Закупоч-
ная_цена, Количество)
SELECT
ПОСТАВКА.Индекс_товара,ПОСТАВКА.Код_поставщика,
ПОСТАВКА.Дата_поступления,
ПОСТАВКА.Закупочная_цена, ПОСТАВКА.Количество
FROM ПОСТАВКА
WHERE ПОСТАВКА.Дата_поступления = #10/09/12#;
```

Оператор удаления данных **DELETE** позволяет удалить одну или несколько строк из таблицы в соответствии с условиями, которые задаются для удаляемых строк. Синтаксис оператора DELETE следующий:

```
DELETE FROM <имя_таблицы>
[WHERE <условия_отбора>]
```

Если условия отбора не задаются, то из таблицы удаляются все строки, однако это не означает, что удаляется вся таблица. Исходная таблица остается, но она будет пустой, незаполненной.

Например, удалить из таблицы R1 данные о сдаче экзамена студентом Антоновым по дисциплине «БД»:

```
DELETE Э.*  
FROM Экзамены Э  
WHERE (ФИО = «Антонов») AND (Дисциплина =  
«БД»);
```

Все операции манипулирования данными связаны с понятием целостности данных. Операции манипулирования данными не всегда выполнимы, даже если синтаксически они написаны правильно, если эти операции нарушают целостность данных.

Операция обновления данных **UPDATE** требуется тогда, когда происходят изменения данных, которые надо отразить в базе данных.

Запрос на обновление может изменить сразу целую группу записей. Этот запрос состоит из трех частей:

- предложение **UPDATE**, которое указывает на обновляемую таблицу;
- предложение **SET**, задающее данные для обновления;
- необязательный критерий **WHERE**, ограничивающий число записей, на которые воздействует запрос на обновление.

Примеры.

Изменить на 3 оценку по дисциплине «БД» у студента Миронова:

```
UPDATE Экзамены SET R1.Оценка = 3  
WHERE (ФИО = «Миронов») AND (Дисциплина = «БД»);
```

Назначить всем стипендию в 2000 рублей:

```
UPDATE Студенты SET Стипендия = 2000;
```

В таблице Сотрудники обновить поле Оклад. Данное поле увеличить на процент, который запрашивается.

```
UPDATE Сотрудники  
SET Оклад = Оклад * (100 + [Введите процент])/100;
```

В таблице Сотрудники обновить поле Оклад у сотрудника, табельный номер которого запрашивается. Поле Оклад увеличивается на процент, который запрашивается.

```
UPDATE Сотрудники
SET Оклад = Оклад * (100 + [Введите процент])/100
WHERE [Табельный номер] = [Введите табельный номер];
```

Даны две таблицы Бензин (Марка, Цена, Общее_количество), Продажа (Чек, Марка, Цена_продажи, Дата, Продано). Обновить поле Цена_продажи в таблице Продажа, присвоить ему значение поля Цена из таблицы Бензин для соответствующих марок бензина.

```
UPDATE Продажа INNER JOIN Бензин
ON Бензин.Марка = Продажа.Марка
SET Продажа.Цена_продажи = Бензин.Цена;
```

3.5.11. Запросы на создание таблиц

Запрос на создание таблиц аналогичен запросу на добавление, за исключением того, что он создает новую таблицу и сразу же копирует в нее записи. Общий вид запроса:

```
SELECT <список полей> INTO<имя новой таблицы>
```

Примеры.

Из таблицы ЗАКАЗЫ сформировать новую таблицу АРХИВ_ЗАКАЗОВ, которая содержит заказы, сделанные до 1 января 2012 года:

```
SELECT ЗАКАЗЫ.* INTO АРХИВ_ЗАКАЗОВ
FROM ЗАКАЗЫ
WHERE [Дата заказа] <= #01/01/12#;
```

Из таблицы ПОСТАВКА сформировать новую таблицу АРХИВ, в которую отправить записи с датой поступления, совпавшей с запрашиваемой:

```
SELECT      ПОСТАВКА.Индекс_товара,      ПОСТАВ-
КА.Код_поставщика,      ПОСТАВКА.Дата_поступления,      ПО-
СТАВКА.Количество INTO АРХИВ FROM ПОСТАВКА
WHERE ПОСТАВКА.Дата_поступления=[Введите дату по-
ступления];
```

3.5.12. Использование языка определения данных

Команды языка определения данных (Data Definition Language – DDL) представляют собой инструкции SQL, которые позволяют создавать и модифицировать элементы структуры базы данных. Например, используя DDL, можно создавать, удалять таблицы и изменять их структуру, создавать и удалять индексы.

Создание таблицы. Оператор создания таблицы имеет следующий вид:

```
CREATE TABLE <имя таблицы>
(<имя столбца> <тип данных> [NOT NULL]
[,<имя столбца> <тип данных> [NOT NULL]]...)
```

Обязательными операндами оператора являются имя создаваемой таблицы и имя хотя бы одного столбца (поля) с указанием типа данных, хранимых в этом столбце.

При создании таблицы для отдельных полей могут указываться некоторые дополнительные правила контроля вводимых в них значений. Например, конструкция NOT NULL (не пустое) служит для определения обязательного поля.

Следует отметить, что в коммерческих продуктах определение типов данных не полностью согласуется с требованиями официального стандарта SQL. Это объясняется желанием обеспечить совместимость SQL с другими языками программирования.

Стандарт SQL поддерживает следующие типы данных.

1. Строка данных

CHARACTER [(длина)] или CHAR [(длина)]

Значения данного типа имеют фиксированную длину, которая определяется параметром длина. Этот параметр может принимать значения от 1 до 255 символов. Если вводимая текстовая константа меньше заданной длины, то автоматически строка дополняется справа пробелами.

Некоторые реализации языка SQL поддерживают в качестве типа данных строки переменной длины. Этот тип обозначается

VARCHAR [(длина)]

Этот тип описывает текстовую строку, которая может иметь произвольную длину. Максимальная длина определяется конкретной реализацией языка SQL (например, в ORACLE – до 2000 символов). При вводе текстовой константы, длина которой меньше заданной, не происходит дополнение ее пробелами до заданного максимального значения.

Текстовые константы в выражениях SQL заключаются в одиночные кавычки.

Если длина строки не указана явно, она полагается равной одному символу во всех случаях.

2. Числовые типы данных.

Стандартными числовыми типами данных SQL являются:

INTEGER – используется для представления целых чисел в диапазоне от -2^{31} до $+2^{31}$;

SMOLLINT – используется для представления чисел в диапазоне от -2^{15} до $+2^{15}$;

DECIMAL (точность [, масштаб]) – десятичное число с фиксированной точкой, точность определяет количество значащих цифр в числе. Масштаб указывает максимальное число цифр справа от точки.

NUMERIC (точность[, масштаб]) – десятичное число с фиксированной точкой, такое же, как и DECIMAL;

FLOAT [(точность)] – число с плавающей точкой и указанной минимальной точностью;

REAL – число такое же, как при типе FLOAT, за исключением определения точности по умолчанию (в зависимости от конкретной реализации SQL);

DOUBLE PRECISION – число аналогично REAL, но точность в два раза выше точности REAL.

3. Дата и время.

Тип данных, предназначенный для представления даты и времени, является нестандартным. Каждая конкретная СУБД специфически определяет эти типы.

Константы типа даты записываются в зависимости от формата, принятого в операционной системе. Например, '03.05.2006', '12/09/2006', '09-ноя-2006', '07-апр-06'.

При создании таблицы возможно задание ограничений.

1. Ограничение Not null может быть установлено для любого поля реляционной таблицы. При наличии ограничения Not null запрещается ввод значений null в это поле. Атрибут null устанавливается по умолчанию. Атрибут Not null означает, что в поле обязательно должна быть внесена информация.

2. Ограничения первичного ключа заключаются в следующем: первичные ключи указываются при создании таблицы, для них обязательно ограничение Not null, которое записывается после определения типа поля. Если ключ составной, для каждого поля надо указывать Not null.

3. Ограничение Unique означает, что все значения в поле уникальны. Однако в отличие от первичного ключа допускается наличие в поле пустых значений (если не установлено Not null). Ограничение записывается после определения типа поля.

4. Значение по умолчанию задается с помощью Default. Оно записывается после определения типа поля, например:

Create table имя_таблицы

(... имя_поля тип_данных Default= значение по умолчанию).

Рассмотрим создание таблиц с использованием различных СУБД.

В табл. 3.3 перечислены типы данных, которые можно использовать при создании таблиц, используя Microsoft Jet DDL и предложение CREATE (СУБД Access).

Типы данных полей, доступных в Jet Таблица 3.3

Тип данных	SQL тип
Счетчик	COUNTER
Текстовый	TEXT
Мемо	LONGTEXT
Денежный	CURRENCY
Дата/время	DATETIME
Числовой (одинарное с плавающей точкой)	SINGLE
Числовой (двойное с плавающей точкой)	DOUBLE
Числовой (целое)	INTEGER
Числовой (длинное целое)	LONG
Числовой (байт)	BYTE

С помощью конструкции **CONSTRAINT** можно задать первичный ключ таблицы.

Пример 1. Создание таблицы TABL1.

```
CREATE TABLE TABL1
([FIL1] COUNTER, [FIL2] TEXT (10),
[FIL3] CURRENCY, [FIL4] DATETIME, [FIL5] BYTE,
[FIL6] INTEGER, [FIL7] SINGLE, [FIL8] LONG,
[FIL9] DOUBLE,
CONSTRAINT PrimaryKey PRIMARY KEY ([FIL1]))
```

В данном примере поле FIL1 объявлено ключевым, для данного поля создан индекс с именем PrimaryKey.

Пример 2. Создание таблицы TABL2.

```
CREATE TABLE TABL2
([FIL1] INTEGER, [FIL2] TEXT (10) NOT NULL,
[FIL3] CURRENCY, [FIL4] LONGTEXT,
```

```
CONSTRAINT PrimaryKey PRIMARY  
KEY ([FIL1],[FIL2]) )
```

В данном примере поля FIL1, FIL2 объявлены ключами и поле FIL2 объявлено также обязательным.

Пример 3. Создание таблицы TABL2 (структура таблицы совпадает с таблицей из примера 2). В данной таблице поле FIL1 объявлено внешним ключом. Между таблицами TABL1 и TABL2 устанавливается связь «один-ко-многим» по полю FIL1.

```
CREATE TABLE TABL2  
([FIL1] INTEGER, [FIL2] TEXT (10) NOT NULL, [FIL3]  
CURRENCY, [FIL4] LONGTEXT,  
CONSTRAINT PrimaryKey PRIMARY KEY  
([FIL1],[FIL2]),  
CONSTRAINT ForeignKey FOREIGN KEY ([FIL1])  
REFERENCES TABL1 ([FIL1]))
```

Примеры создания таблиц с использованием языка SQL, встроенного в Delphi.

Пример 1. Создание таблицы STUDENT.

```
CREATE TABLE STUDENT  
(Nom_z Character (10), Gr Character (10), Fam Varchar (30)  
Not null)
```

Пример 2. Создание таблицы STUDENT с ключевым полем Nom_z

```
CREATE TABLE STUDENT  
(Nom_z Character (10) Not null primary key, Gr Varchar (10),  
Fam Varchar (30) not null)
```

Пример 3. Создания таблицы STUDENT с двумя ключевыми полями

```
CREATE TABLE STUDENT  
(Nom_z Character (10) Not null, Gr Varchar (10) Not null,  
Fam Varchar (30), primary key (Nom_z, Gr))
```

Пример 4. Создание таблиц и связывание их между собой. Таблицы:

Firma (Nomer, Naim, Adres), Tovar (Kod, Nazv, Cena),

Postavka (Nomer_p, Nomer, Kod, Kol).

Первая и третья таблицы связаны по полю Nomer, вторая и третья таблицы связаны по полю Kod.

Create table Firma

(Nomer Integer Not null primary key, Naim Varchar (150), Adres Varchar (50))

Create table Tovar

(Kod Integer Not null primary key, Nazv Varchar (50), Cena decimal (8,2))

Create table Postavka

(Nomer_p Integer Not null primary key, Nomer Integer, Kod Integer, Kol Decimal (8,2),

foreign key Firma_Postavka (Nomer) references Firma (Nomer), foreign key Tovar_Postavka (Kod) references Tovar (Kod))

Удаление таблиц. Удалять элементы базы данных можно с помощью предложения **DROP**. Оператор удаления таблиц имеет следующий вид:

DROP TABLE <имя таблицы>

Например, удаление таблицы TABL2 можно осуществить следующим образом:

DROP TABLE TABL2

Модификация структуры таблицы. Оператор **ALTER TABLE** изменения структуры таблицы имеет следующий вид:

ALTER TABLE <имя таблицы>

MODIFY | ADD | DROP <имя поля> [<тип данных>]

Пример 1. Добавление поля.

ALTER TABLE Student

ADD (Email Character (25))

Пример 2. Изменение типа поля.

```
ALTER TABLE Student  
MODIFY (Email Varchar (25))
```

Пример 3. Удаление столбца.

```
ALTER TABLE Student  
DROP (Email)
```

Создание индексов. Помимо создания индексов в процессе формирования таблицы (с помощью предложения **CONSTRAINT**), можно также создавать индексы уже после того, как таблица сформирована (с помощью предложения **CREATE INDEX**). Это приносит эффект в тех случаях, когда таблица уже существует (в то время как конструкция **CONSTRAINT** применяется для формирования индексов только в момент создания таблицы). Оператор создания индекса имеет следующий вид:

```
CREATE [UNIQUE] INDEX <имя индекса>  
ON <имя таблицы>  
(<имя столбца> [ASC | DESC]  
[, <имя столбца> [ASC | DESC]...)
```

Оператор позволяет создать индекс для одного или нескольких столбцов заданной таблицы с целью ускорения выполнения запросных и поисковых операций с таблицей. Для одной таблицы можно создать несколько индексов.

Задав необязательную опцию **UNIQUE**, можно обеспечить уникальность значений во всех указанных в операторе столбцах. По существу, создание индекса с указанием признака **UNIQUE** означает определение ключа в созданной ранее таблице.

При создании индекса можно задать порядок автоматической сортировки значений в столбцах – в порядке возрастания **ASC** (по умолчанию), или в порядке убывания **DESC**. Для разных столбцов можно задавать различный порядок сортировки.

Пример 1: создание индекса FIL2_Index в таблице TABL1 по полю FIL2.

```
CREATE INDEX FIL2_Index  
ON TABL1 ([FIL2])
```

Пример 2: создание уникального индекса FIL1_Index (имеет разные значения, в том числе и NULL) по полю FIL1 в таблице TABL1.

```
CREATE UNIQUE INDEX FIL1_Index  
ON TABL1 ([FIL1])
```

Пример 3: создание уникального индекса FIL3_Index, не имеющего значения NULL, по полю FIL3 в таблице TABL1.

```
CREATE UNIQUE INDEX FIL3_Index  
ON TABL1 ([FIL3])  
WITH DISALLOW NULL
```

Пример 4: добавление первичного ключа в таблицу TABL1 и построение для него индекса PrimaryKey. Ключом объявляется поле FIL1.

```
CREATE UNIQUE INDEX PrimaryKey  
ON TABL1 ([FIL1])  
WITH PRIMARY
```

Пример 5: создание в таблице TABL1 индекса для двух полей, одно из которых сортируется по возрастанию (FIL2), а второе поле сортируется по убыванию (FIL3).

```
CREATE INDEX FIL2_FIL3_Index  
ON TABL1 ([FIL2],[FIL3] DESC)
```

Удаление индекса. Оператор удаления индекса **DROP INDEX** имеет следующий вид:

```
DROP INDEX <имя индекса> ON <имя таблицы>
```

Пример: удаление индекса FIL3_Index из таблицы TABL1.

```
DROP INDEX FIL3_Index ON TABL1
```

4. ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

4.1. Этапы проектирования БД

Проектирование базы данных является одним из важнейших **этапов жизненного цикла БД**, который включает:

- проектирование БД,
- проектирование приложений,
- реализацию БД,
- разработку специальных средств администрирования БД,
- эксплуатацию БД.

Процесс проектирования БД представляет собой последовательность переходов от неформального словесного описания информационной структуры предметной области к формализованному описанию объектов предметной области в терминах некоторой модели.

Можно выделить следующие **этапы проектирования базы данных**.

- системный анализ предметной области;
- инфологическое проектирование;
- выбор СУБД;
- даталогическое проектирование (логическое проектирование).

В рамках **системного анализа предметной области** необходимо провести подробное словесное описание предметной области и реальных связей, которые существуют между описываемыми объектами.

В общем случае существуют два подхода к выбору состава и структуры предметной области.

Функциональный подход. Применяется тогда, когда заранее известны задачи, для решения которых создается база данных. В этом случае можно четко выделить минимально необходимый набор объектов предметной области, которые должны быть описаны.

Предметный подход. Информационные потребности будущих пользователей БД жестко не фиксированы. Невозможно

точно выделить минимальный набор объектов предметной области, которые необходимо описать. В этом случае в описание предметной области включаются такие объекты и взаимосвязи, которые наиболее характерны и наиболее существенны для нее. БД, конструируемая при этом, называется предметной, то есть она может быть использована при решении множества разнообразных, заранее не определенных задач.

Чаще всего на практике рекомендуется использовать некоторый компромиссный вариант, который, с одной стороны, ориентирован на конкретные задачи или функциональные потребности пользователей, а с другой стороны, учитывает возможность наращивания новых приложений.

Системный анализ должен заканчиваться техническим заданием на проектирование базы данных и приложения для работы с ней:

- подробным описанием объектов предметной области, информацию о которых требуется хранить в БД;
- формулировкой конкретных задач, которые будут решаться с использованием данной БД, с кратким описанием алгоритмов их решения;
- описанием выходных документов, которые должны генерироваться в системе,
- описанием входных документов, которые служат основанием для заполнения данными БД.

Инфологическое проектирование создает инфологическую модель предметной области, не привязанную к конкретной СУБД.

Инфологическое проектирование, прежде всего, связано с попыткой представления семантики предметной области в модели БД. Информационная модель предметной области представляет собой схему базы данных, содержащую описание структуры таблиц (имена, типы, размеры полей, ключевые поля) и связей между ними.

Логическое проектирование приводит к разработке схемы БД с учетом особенностей выбранной СУБД. Схема реляци-

онной базы данных есть совокупность схем отношений, которые адекватно моделируют объекты предметной области, их основные свойства и связи между объектами.

Логическое проектирование заключается:

- в определении числа и структуры таблиц, описываемых средствами выбранной СУБД;
- в описании запросов к базе данных;
- в описании выходных документов;
- в разработке алгоритмов обработки информации;
- в проектировании форм для ввода и редактирования данных в таблицах базы данных.

Часто этапы инфологическое проектирование, выбор СУБД и логическое проектирование объединяют и называют логическим проектированием.

Решение проблем проектирования на **физическом уровне** во многом зависит от используемой СУБД, зачастую автоматизировано и скрыто от пользователя. В ряде случаев пользователю предоставляется возможность настройки отдельных параметров системы.

4.2. Проблемы проектирования реляционных баз данных

Рассмотрим основные проблемы, имеющие место при определении структур данных в отношениях реляционной модели.

Избыточное дублирование данных. Следует различать простое (неизбыточное) и избыточное дублирование данных. Наличие первого из них допускается в базах данных, а избыточное дублирование данных может приводить к проблемам при обработке данных. Пример неизбыточного дублирования данных приведен на рис 4.1. Для сотрудников, находящихся в одном помещении, номера телефонов совпадают. Однако для каждого служащего номер телефона уникален. Поэтому ни один из номеров не является избыточным. Действительно, при удалении одного из номеров телефонов будет утеряна инфор-

мация о том, по какому номеру можно дозвониться до одного из служащих

Сотрудники_Телефоны	
Фамилия	Телефон
Иванов	16-30-67
Петров	16-30-45
Сидоров	16-30-45
Егоров	16-30-45
Сергеев	16-30-23

Рис. 4.1. Неизбыточное дублирование

Пример избыточного дублирования представлен на рис. 4.2, где приведено отношение, в которое дополнено атрибутом Номер_комнаты. Естественно предположить, что все служащие в одной комнате имеют один и тот же телефон. В рассматриваемом отношении имеется избыточное дублирование данных. Номера телефонов Петрова и Сидорова, например, можно узнать из кортежа со сведениями о Егорове.

Сотрудники_Телефоны_Комнаты		
Фамилия	Телефон	Комнаты
Иванов	16-30-67	109
Петров	16-30-45	112
Сидоров	16-30-45	112
Егоров	16-30-45	112
Сергеев	16-30-23	120

Рис. 4.2. Избыточное дублирование

На рис. 4.3. приведен пример неудачного отношения Сотрудники_Телефоны_Комнаты приведен пример отношения, в котором вместо телефонов Сидорова и Егорова поставлены прочерки (неопределенные значения).

Сотрудники_Телефоны_Комнаты

Фамилия	Телефон	Комнаты
Иванов	16-30-67	109
Петров	16-30-45	112
Сидоров	-	112
Егоров	-	112
Сергеев	16-30-23	120

Рис. 4.3. Избыточное дублирование

Неудачность подобного способа исключения избыточности заключается в следующем. Во-первых, при программировании придется потратить дополнительные усилия на создание механизма поиска информации для прочерков таблицы. Во-вторых, память все равно выделяется под атрибуты с прочерками, хотя дублирование данных и исключено. В-третьих, что особенно важно, при исключении из коллектива Петрова кортеж со сведениями о нем будет исключен из отношения, а значит уничтожена информация о телефоне 112-й комнаты, что недопустимо.

Возможный способ выхода из данной ситуации приведен на рис. 4.4.

Сотрудники Комнаты		Телефоны Комнаты	
Фамилия	Комнаты	Телефон	Комнаты
Иванов	109	16-30-67	109
Петров	112	16-30-45	112
Сидоров	112	16-30-23	120
Егоров	112		
Сергеев	120		

Рис. 4.4. Исключение избыточного дублирования

Здесь показаны два отношения, полученные путем декомпозиции исходного отношения. Первое из них содержит ин-

формацию о сотрудниках и номерах комнат, где они работают, а второе – информацию о номерах телефонов в каждой из комнат. Теперь, если Петров уволится из учреждения и, как следствие этого, будет удалена всякая информация о нем из базы данных, это не приведет к утере информации о номере телефона в 112-й комнате.

Процедура декомпозиции исходного отношения на два новых является основной процедурой нормализации отношений.

Аномалии. Избыточное дублирование данных создает проблемы при обработке кортежей отношения, названные Э. Коддом «аномалиями обновления отношения». Он показал, что для некоторых отношений проблемы возникают при попытке удаления, добавления или редактирования их кортежей.

Аномалиями будем называть такую ситуацию в таблицах БД, которая приводит к противоречиям в БД либо существенно усложняет обработку данных.

Выделяют три основных вида аномалий: аномалии модификации (или редактирования), аномалии удаления и аномалии добавления.

Аномалии модификации проявляются в том, что изменение значения одного данного может повлечь за собой просмотр всей таблицы и соответствующее изменение некоторых других записей таблицы. Например, изменение номера телефона в комнате 112 (рис.4.2), что представляет собой один единственный факт, потребует просмотра всей таблицы Сотрудники_Телефоны_Комнаты и изменения поля Телефон в записях, относящихся к Петрову, Сидорову и Егорову.

Аномалии удаления состоят в том, что при удалении какого-либо данного из таблицы может пропасть и другая информация, которая не связана напрямую с удаляемым данным. В той же таблице удаление записи о сотруднике Иванове приводит к исчезновению информации о номере телефона, установленного в 109-й комнате.

Аномалии добавления возникают в случаях, когда информацию в таблицу нельзя поместить до тех пор, пока она неполная, либо вставка новой записи требует дополнительного просмотра таблицы. Примером может служить операция добавления нового сотрудника в таблицу на рис. 4.2. Очевидно, будет противоестественным хранение сведений в этой таблице только о комнате и номере телефона в ней, пока никто из сотрудников не помещен в нее. Более того, если в данной таблице поле Фамилия является ключевым, то хранение в ней неполных записей с отсутствующей фамилией служащего просто недопустимо из-за неопределенности значения ключевого поля.

Вторым примером возникновения аномалии добавления может быть ситуация включения в таблицу нового сотрудника. При добавлении таких записей для исключения противоречий желательно проверить номер телефона и соответствующий номер комнаты хотя бы с одним из сотрудников, сидящих с новым сотрудником в той же комнате. Если же окажется, что у некоторых сотрудников, сидящих в одной комнате, имеются разные телефоны, то вообще не ясно, что делать (то ли в комнате несколько телефонов, то ли какой-то из номеров ошибочный).

4.3. Нормализация отношений

Проектирование БД является одним из этапов жизненного цикла информационной системы. Основной задачей, решаемой в процессе проектирования БД, является задача нормализации отношений. Рассматриваемый ниже метод нормальных форм является классическим методом проектирования реляционных БД. Этот метод основан на фундаментальном в теории реляционных баз данных понятии зависимости между атрибутами отношений.

Исследование зависимостей позволяет грамотно проектировать схемы баз данных, получать отношения, обладающие необходимыми свойствами.

Состав атрибутов отношения должен удовлетворять двум основным требованиям: между атрибутами не должно быть нежелательных функциональных зависимостей; группировка атрибутов должна обеспечивать минимальное дублирование данных, их обработку и обновление без трудностей.

Удовлетворение этих требований достигается нормализацией отношений базы данных.

Нормализация отношений – это пошаговый обратимый процесс декомпозиции (разложения) исходных отношений базы данных на другие, более мелкие, с целью ликвидации нежелательных функциональных зависимостей.

Нормализация отношений позволяет устранить избыточное дублирование данных, обеспечивает их непротиворечивость, уменьшает трудозатраты на ведение (ввод, корректировку, удаление) базы данных.

Аппарат нормализации отношений был разработан Е. Коддом. В нем определяются различные нормальные формы. Каждая нормальная форма ограничивает типы допустимых функциональных зависимостей отношения и устраняет некоторые аномалии при выполнении операций над отношениями базы данных.

Функциональные зависимости. Дадим определение функциональной зависимости.

Функциональная зависимость – один из возможных типов зависимостей между атрибутами.

Пусть X и Y – атрибуты отношения R . Атрибут Y отношения R функционально зависит от атрибута X отношения R , если в каждый момент времени каждому значению X соответствует одно и то же значение Y . Функциональная зависимость записывается: $X \rightarrow Y$.

Если $X \rightarrow Y$ и $Y \rightarrow X$, то между X и Y существует взаимно однозначное соответствие.

Атрибуты X и Y могут быть составными, т.е. могут представлять собой группы, состоящие из двух и более атрибутов. С практической точки зрения смысл данного определения состоит

в том, что если Y функционально зависит от X , то каждый из кортежей, имеющих одно и то же значение X , должен иметь также одно и то же значение Y . Значения X и Y могут изменяться время от времени, но при этом должны изменяться так, чтобы каждое уникальное значение X имело только одно уникальное значение Y , связанное с ним.

В конкретной ситуации функциональные зависимости определяются путем детализации свойств всех атрибутов в отношении, т.е. функциональные зависимости получают исходя из природных свойств самих атрибутов.

Нормальные формы. Процесс проектирования БД с использованием метода нормальных форм является итерационным и заключается в последовательном переводе отношений из первой нормальной формы в нормальные формы более высокого порядка по определенным правилам. Каждая следующая нормальная форма ограничивает определенный тип функциональных зависимостей, устраняет соответствующие аномалии при выполнении операций над отношениями БД и сохраняет свойства предшествующих нормальных форм.

Выделяют следующие нормальные формы:

- первая нормальная форма (1НФ);
- вторая нормальная форма (2НФ);
- третья нормальная форма (3НФ);
- усиленная третья нормальная форма, или нормальная форма Бойса-Кодда (БКНФ);
- четвертая нормальная форма (4НФ);
- пятая нормальная форма (5НФ).

Отношение находится в **первой нормальной форме (1НФ)** тогда и только тогда, когда все входящие в него атрибуты являются атомарными, т.е. значения атрибутов, образующие соответствующие домены, рассматриваются как неделимые, а не как множества из более элементарных доменов. Неделимость поля означает, что содержащиеся в нем значения не должны делиться на более мелкие или представлять из себя повторяющие-

ся группы. В общем случае можно считать, что отношение и отношение в первой нормальной форме – это синонимы.

Отношение находится во **второй нормальной форме (2НФ)**, если оно находится в 1НФ и каждый неключевой атрибут функционально полно зависит от первичного (составного) ключа.

Таким образом, если в отношении нет частичных функциональных зависимостей, то оно находится в 2НФ.

Дадим определение частичной и полной функциональным зависимостям.

Если в отношении неключевой атрибут функционально зависит от части составного ключа, то между ключом и неключевым атрибутом имеет место **частичная функциональная зависимость**.

Если в отношении неключевой атрибут функционально зависит от составного ключа и не находится в функциональной зависимости от любых его частей, то неключевой атрибут **функционально полно зависит** от ключа.

Рассмотрим отношение, в котором присутствуют частичные и полные функциональные зависимости:

Выдача книг (Шифр книги, Авторы, Название, Год издания, Номер читательского билета, Ф.и.о., Шифр группы, Курс, Специальность, Факультет, Вид обучения, Дата выдачи, Дата возврата).

Ключом данного отношения являются поля: Шифр книги, Номер читательского билета.

Полная функциональная зависимость:

Шифр книги, Номер читательского билета -> Дата выдачи, Дата возврата.

Частичные функциональные зависимости:

Шифр книги -> Авторы, Название, Год издания.

Номер читательского билета -> Ф.и.о., Шифр группы, Курс, Специальность, Факультет, Вид обучения.

Частичные функциональные зависимости приводят к аномалиям:

избыточно дублируются сведения о книгах (авторы, название, год издания), поскольку одинаковую книгу могут взять несколько читателей;

избыточно дублируются сведения о читателях (ф.и.о., шифр группы, курс и т.д.), поскольку читатель может взять несколько книг;

существует проблема контроля правильности данных: измененные сведения о книге или читателе приходится вносить более чем в один кортеж;

существует проблема добавления записей: включить в таблицу кортеж о новой книге можно только, если эта книга взята; включить в таблицу кортеж о новом читателе можно только, если он взял книгу;

существует проблема удаления записей: удаление кортежа может привести к потере сведений о книге или читателе.

Перевод отношения в следующую нормальную форму осуществляется методом «декомпозиции без потерь». Такая декомпозиция должна обеспечить то, что запросы (выборка данных по условию) к исходному отношению и к отношениям, получаемым в результате декомпозиции, дадут одинаковый результат. Основной операцией метода является операция проекции.

Чтобы устранить частичную зависимость и привести отношение во 2НФ, необходимо разложить его на новые отношения следующим образом:

построить проекцию без атрибутов, которые находятся в частичной зависимости от составного ключа;

построить проекцию на часть составного ключа и атрибуты, зависящие от этой части.

При разложении отношения Выдача книг получают три новых отношения:

Выдача книг (Шифр книги, Номер читательского билета, Дата выдачи, Дата возврата);

Книги (Шифр книги, Авторы, Название, Год издания);

Читатели (Номер читательского билета, Ф.и.о., Шифр группы, Курс, Специальность, Факультет, Вид обучения).

В первом отношении ключом являются атрибуты – Шифр книги, Номер читательского билета; во втором – Шифр книги; в третьем – Номер читательского билета.

Отношение находится в **третьей нормальной форме (3НФ)**, если оно находится в 2НФ и в нем отсутствуют транзитивные зависимости неключевых атрибутов от ключа.

Если в отношении для атрибутов X, Y, Z имеют место функциональные зависимости $X \rightarrow Y$, $Y \rightarrow Z$, но обратная зависимость отсутствует, то атрибут Z **транзитивно** зависит от атрибута X .

Рассмотрим отношение, в котором присутствуют транзитивные зависимости:

Сотрудники (Табельный номер, Ф.и.о., Кафедра, Номер корпуса, Номер аудитории, Телефон кафедры, Должность, Категория, Оклад, Надбавка за должность).

Ключом данного отношения является Табельный номер.

В отношении Сотрудники имеют место следующие функциональные зависимости:

Табельный номер $\rightarrow$ Ф.и.о., Кафедра, Номер корпуса, Номер аудитории, Телефон кафедры, Должность, Категория, Оклад, Надбавка за должность;

Кафедра $\rightarrow$ Номер корпуса, Номер аудитории, Телефон кафедры;

Должности $\rightarrow$ Надбавка за должности;

Категория $\rightarrow$ Оклад.

Три последние зависимости являются транзитивными.

Наличие транзитивных зависимостей порождает аномалии следующего характера:

избыточно дублируется информация о кафедрах, должностях, категориях;

сложное редактирование данных: изменение, например, оклада требует поиска и замены его во всех кортежах для сотрудников, имеющих одинаковую категорию;

существует проблема удаления записей: удаление кортежа может привести к потере сведений о кафедре, должности, категории;

существует проблема добавления записей: невозможно, например, включить сведения о новой должности, если эту должность никто не занимает.

Чтобы устранить транзитивную зависимость и привести отношение в 3НФ, необходимо:

построить проекцию без атрибутов, находящихся в транзитивной зависимости от ключа;

построить проекцию на неключевые атрибуты, между которыми имеется функциональная зависимость.

В нашем примере получим:

Сотрудники (Табельный номер, Ф.и.о., Кафедра, Должность, Категория);

Кафедры (Кафедра, Номер корпуса, Номер аудитории, Номер телефона);

Должности (Должность, Надбавка за должность);

Категории (Категория, Оклад).

В первом отношении ключ – Табельный номер, во втором – Кафедра, в третьем – Должность, в четвертом – Категория.

На практике процесс проектирования схем отношений заканчивается, когда отношения приведены к 3НФ.

Усиленная 3НФ или нормальная форма Бойса-Кодда (БКНФ). Если в отношении имеется зависимость атрибутов составного ключа от неключевых атрибутов, то необходимо перейти к усиленной 3НФ.

Отношение находится в БКНФ, если оно находится в 3НФ и в нем отсутствуют зависимости ключей (атрибутов составного ключа) от неключевых атрибутов.

Можно дать другое определение усиленной 3НФ.

Отношение находится в нормальной форме Бойса-Кодда, если оно находится в третьей нормальной форме и каждый детерминант отношения является возможным ключом отношения.

Если в отношении существует несколько функциональных зависимостей, то каждый атрибут или набор атрибутов, от которого зависит другой атрибут, называется *детерминантом отношения*.

Рассмотрим отношение, моделирующее сдачу текущих экзаменов. Предположим, что студент может сдавать экзамен по одной дисциплине несколько раз, если он получил неудовлетворительную оценку. Допустим, что студента можно однозначно идентифицировать номером его зачетной книги, в то же время для электронного учета в период обучения каждый студент получает уникальный номер-идентификатор. Отношение, которое моделирует сдачу текущей сессии, имеет следующую структуру:

Экзамены(Номер зач_кн, Идентификатор_студента, Дисциплина, Дата, Оценка)

Возможными ключами отношения являются Номер зач_кн, Дисциплина, Дата и Идентификатор-студента, Дисциплина, Дата.

В данном отношении имеют место следующие функциональные зависимости:

Номер зач_кн, Дисциплина, Дата -> Оценка;

Идентификатор_студента, Дисциплина, Дата -> Оценка;

Номер зач_кн -> Идентификатор_студента;

Идентификатор_студента -> Номер зач_кн.

Это отношение находится в третьей нормальной форме, потому что в нем отсутствуют частичные и транзитивные зависимости.

Для приведения отношения к нормальной форме Бойса-Кодда надо разделить отношение, например, на два со следующими схемами:

Экзамен (Идентификатор_студента, Дисциплина, Дата, Оценка)

Идентификаторы (Номер зач_кн, Идентификатор_студента)

В большинстве случаев достижение третьей нормальной формы или даже формы Бойса-Кодда считается достаточным для реальных проектов баз данных, однако в теории нормализации существуют нормальные формы высших порядков, которые уже связаны не с функциональными зависимостями между атрибутами отношений, а отражают более тонкие вопросы семантики предметной области и связаны с другими видами зависимостей.

Дадим несколько определений.

В отношении $R(A, B, C)$ существует **многозначная зависимость** $A \twoheadrightarrow B$ в том и только в том случае, если множество значений B , соответствующее паре значений A и C , зависит только от A и не зависит от C .

При рассмотрении функциональных зависимостей каждому значению детерминанта соответствовало только одно значение зависимого от него атрибута. При рассмотрении многозначных зависимостей одному значению некоторого атрибута соответствует устойчиво постоянное множество значений другого атрибута.

Пусть дано отношение, которое моделирует предстоящую сдачу экзаменов на сессии. Допустим, оно имеет вид:

(Номер зач_кн, Группа, Дисциплина)

Перечень дисциплин, которые должен сдавать студент, однозначно определяется не его фамилией, а номером группы (то есть специальностью, на которой он учится).

В данном отношении существуют две многозначные зависимости:

Группа $\twoheadrightarrow$ Дисциплина,

Группа $\twoheadrightarrow$ Номер зач_кн.

Это означает, что каждой группе однозначно соответствует перечень дисциплин по учебному плану и номер группы определяет список студентов, которые в этой группе учатся.

Если работать с исходным отношением, то невозможно внести информацию о новой группе и ее учебном плане (перечне дисциплин, которые должна изучить группа) до тех пор, пока

в нее не будут зачислены студенты. При изменении перечня дисциплин по учебному плану, например при добавлении новой дисциплины, внести эти изменения в отношении для всех студентов, занимающихся в данной группе, весьма затруднительно. С другой стороны, если мы добавляем студента в уже существующую группу, то мы должны добавить множество кортежей, соответствующих перечню дисциплин для данной группы. Эти аномалии модификации отношения связаны с наличием двух многозначных зависимостей.

В теории реляционных баз данных доказывается, что в общем случае в отношении $R(A,B,C)$ существует многозначная зависимость $A \twoheadrightarrow B$ в том и только в том случае, когда существует многозначная зависимость $A \twoheadrightarrow C$.

Дальнейшая нормализация отношений основывается на теореме Фейджина.

Теорема Фейджина. Отношение $R(A,B,C)$ можно спроецировать без потерь в отношения $R_1(A,B)$ и $R_2(A,C)$ в том и только в том случае, когда существуют многозначные зависимости $A \twoheadrightarrow B$ и $A \twoheadrightarrow C$.

Под проецированием без потерь понимается такой способ декомпозиции отношения путем применения операции проекции, при котором исходное отношение полностью и без избыточности восстанавливается путем естественного соединения полученных отношений. Практически теорема доказывает наличие эквивалентной схемы для отношения, в котором существует несколько многозначных зависимостей.

Отношение R находится в **четвертой нормальной форме (4НФ)** в том и только в том случае, если в случае существования многозначной зависимости $A \twoheadrightarrow B$ все остальные атрибуты R функционально зависят от A .

В примере можно произвести декомпозицию исходного отношения в два отношения:

(Номер зач_кн, Группа)

(Группа, Дисциплина)

Оба эти отношения находятся в 4НФ и свободны от отмеченных аномалий. Действительно, обе операции модификации теперь упрощаются: добавление нового студента связано с добавлением всего одного кортежа в первое отношение, а добавление новой дисциплины приводит к добавлению одного кортежа во второе отношение, кроме того, во втором отношении мы можем хранить любое количество групп с определенным перечнем дисциплин, в которые пока еще не зачислены студенты.

Следующие нормальные формы в данном пособии не рассматриваются.

Итак, нормализация отношений выполняется декомпозицией (разложением) их схем. Декомпозиция должна гарантировать обратимость, т.е. получение исходных отношений путем выполнения операций соединения над их проекциями.

Процесс нормализации отношений последовательно устраняет частичные зависимости неключевых атрибутов от ключа, транзитивные зависимости неключевых атрибутов от ключа, зависимости ключей от неключевых атрибутов.

Можно сформулировать набор правил, соблюдение которых позволит выполнять требования нормализации:

каждый информационный объект (реляционная таблица) должен содержать уникальный идентификатор (ключ); ключ является простым, если он состоит из одного реквизита, или составным, если из нескольких;

все описательные реквизиты должны быть взаимонезависимы, т.е. между ними не должно быть функциональных зависимостей;

все реквизиты, входящие в составной ключ, должны быть также взаимонезависимы;

каждый описательный реквизит должен функционально полно зависеть от ключа, т.е. каждому значению ключа должно соответствовать только одно значение описательного реквизита;

при составном ключе описательные реквизиты должны зависеть целиком от всей совокупности реквизитов, образующих ключ;

каждый описательный реквизит не должен зависеть от ключа транзитивно, т.е. через другой промежуточный реквизит.

Выполнение требований нормализации обеспечивает построение реляционной базы данных без дублирования данных и возможность поддержания целостности данных при внесении в них изменений.

4.4. Метод сущность-связь

Метод сущность-связь называют также методом «ER-диаграмм». Аббревиатура ER происходит от слов Essence (сущность) и Relation (связь). Метод основан на использовании диаграмм, называемых диаграммами ER-экземпляров и диаграммами ER-типа.

Основные понятия метода. Основными понятиями метода сущность-связь являются следующие:

- сущность;
- атрибут сущности;
- ключ сущности;
- связь между сущностями;
- степень связи;
- класс принадлежности экземпляров сущности;
- диаграммы ER-экземпляров;
- диаграммы ER-типа.

Сущность представляет собой объект, информация о котором хранится в БД. Экземпляры сущности отличаются друг от друга и однозначно идентифицируются. Названиями сущностей являются, как правило, существительные, например: ПРЕПОДАВАТЕЛЬ, ДИСЦИПЛИНА, КАФЕДРА, ГРУППА.

Атрибут представляет собой свойство сущности. Это понятие аналогично понятию атрибута в отношении. Так атрибу-

тами сущности ГРУППА могут быть Шифр группы, Факультет, Специальность, Курс, Фамилия_старосты и т.д.

Ключ сущности – атрибут или набор атрибутов, используемый для идентификации экземпляра сущности. Понятие ключа сущности аналогично определению ключа отношения.

Связь двух или более сущностей предполагает существование зависимостей между атрибутами этих сущностей. Название связи обычно представляется глаголом. Примерами связей между сущностями являются следующие: Преподаватель Ведет Дисциплину (Иванов Ведет «Моделирование»), Преподаватель Преподает в Группе (Иванов Преподает в Группе ВМ-021), Преподаватель Работает на Кафедре (Иванов Работает на Кафедре АВС).

С целью повышения наглядности и удобства проектирования для представления сущностей, экземпляров сущностей и связей между ними используются **диаграммы ER-экземпляров**. Данные диаграммы позволяют оценить степень связи и класс принадлежности сущности.

На рис. 4.5 приведена диаграмма ER-экземпляров для сущностей ПРЕПОДАВАТЕЛЬ и ДИСЦИПЛИНА со связью ВЕДЕТ.

ПРЕПОДАВАТЕЛЬ	ВЕДЕТ	ДИСЦИПЛИНА
СЕРГЕЕВА ●		●МОДЕЛИРОВАНИЕ
ТЮРИН ●		● ЭЛЕКТРОНИКА
БУРКОВСКИЙ ●		● БАЗЫ ДАННЫХ
ПОПОВ ●		● СХЕМОТЕХНИКА
ПИТОЛИН ●		● МЕНЕДЖМЕНТ

Рис.4.5. Диаграмма ER-экземпляров

Диаграмма ER-экземпляров показывает, какую конкретно дисциплину ведет каждый из преподавателей. В приведенной на рис.4.5 диаграмме степень связи между сущностями 1:1 (один преподаватель ведет одну дисциплину и каждая дисцип-

лина читается одним преподавателем), а класс принадлежности обеих сущностей обязательный (все преподаватели и дисциплины участвуют в объявленной связи).

Степень связи является характеристикой связи между сущностями, которая может быть следующей: 1:1, 1:M, M:1, M:M.

Класс принадлежности (КП) сущности может быть: обязательным и необязательным.

Класс принадлежности сущности является **обязательным**, если все экземпляры этой сущности обязательно участвуют в рассматриваемой связи, в противном случае класс принадлежности сущности является **необязательным**.

Варьируя классом принадлежности сущностей для каждого из названных типов связи, можно получить несколько вариантов диаграмм ER-типа.

В диаграмме ER-типа используются следующие обозначения:

- сущность изображают прямоугольником, внутри него записывают имя сущности (существительное);

- под сущностью записывают ключевые поля через запятую, затем ставят запятую и многоточие (означает наличие не ключевых описательных полей); ключевые поля подчеркивают;

- связь между сущностями обозначают ромбом, в котором записывают имя связи (глагол); сущности исвязь соединяют линиями;

- степень связи изображают цифрами или буквами;

- если класс принадлежности обязательный, то точку ставят внутри прямоугольника; если класс принадлежности необязательный, то точку ставят на линии.

Рассмотрим примеры диаграмм ER-типа.

Пример 1.

Связи типа 1:1 и обязательный класс принадлежности.

На рис. 4.6 приведены диаграммы, у которых степень связи между сущностями 1:1, а класс принадлежности обеих сущностей обязательный.

а) диаграмма ER-экземпляров

ПРЕПОДАВА- ТЕЛЬ	ВЕДЕТ	ДИСЦИПЛИНА
СЕРГЕЕВА		МОДЕЛИРОВАНИЕ
ТЮРИН		ЭЛЕКТРОНИКА
БУРКОВСКИЙ		БАЗЫ ДАННЫХ
ПОПОВ		СХЕМОТЕХНИКА
ПИТОЛИН		МЕНЕДЖМЕНТ

б) диаграмма ER-типа



Рис.4.6. Диаграммы для связи 1:1 и обязательным классом принадлежности обеих сущностей

Пример 2. Связи типа 1:M, класс принадлежности первой сущности необязательный, а второй – обязательный. Например, каждый сотрудник кафедры может вести несколько дисциплин или ни одной, но каждая дисциплина ведется одним сотрудником кафедры (рис. 4.7).

а) диаграмма ER-экземпляров

СОТРУДНИКИ	ВЕ- ДЕТ	ДИСЦИПЛИНА
		● ИНФОРМАТИКА
СЕРГЕЕВА	●	● МОДЕЛИРОВАНИЕ
ТЮРИН	●	● ЭЛЕКТРОНИКА
БУРКОВСКИЙ	●	● БАЗЫ ДАННЫХ
ПОПОВ	●	● СХЕМОТЕХНИКА
ПИТОЛИН	●	● МЕНЕДЖМЕНТ
ВАСИЛЬЕВА	●	● ЭЛЕКТРОТЕХНИКА
ПЛОТНИКОВА	●	● ТЕОРИЯ АВТОМАТОВ

б) диаграмма ER-типов

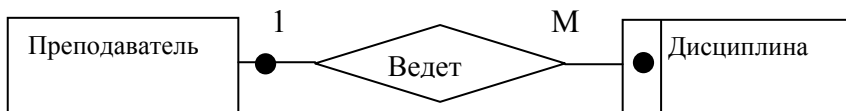


Рис. 4.7. Диаграммы для связи типа 1:М варианта Н-О

Этапы реализации метода сущность-связь. Процесс проектирования базы данных является итерационным – допускающим возврат к предыдущим этапам для пересмотра ранее принятых решений и включает следующие этапы.

1. Выделение сущностей и связей между ними.
2. Построение диаграмм ER-типа с учетом всех сущностей и их связей.
3. Формирование набора предварительных отношений с указанием предполагаемого первичного ключа для каждого отношения.
4. Добавление неключевых атрибутов в отношения.

5. Приведение предварительных отношений к нормальной форме Бойса-Кодда, например, с помощью метода нормальных форм.

6. Пересмотр ER-диаграмм в следующих случаях:

- некоторые отношения не приводятся к нормальной форме Бойса-Кодда;
- некоторым атрибутам не находится логически обоснованных мест в предварительных отношениях.

После преобразования ER-диаграмм осуществляется повторное выполнение предыдущих этапов проектирования (возврат к этапу 1).

Правила формирования отношений. Одним из основных этапов проектирования является этап формирования отношений, который можно формализовать.

Правила формирования отношений основываются на учете следующего:

- степени связи между сущностями (1:1, 1:M, M:1, M:M);
- класса принадлежности экземпляров сущностей (обязательный и необязательный).

Рассмотрим шесть правил формирования отношений на основе диаграмм ER-типа. Правила распространяются на связи между двумя сущностями, т.е. на бинарные связи.

Формирование отношений для связи 1:1.

Правило 1. Если степень бинарной связи 1:1 и класс принадлежности обеих сущностей обязательный, то формируется одно отношение. Первичным ключом этого отношения может быть ключ любого из двух сущностей.

На рис. 4.8 приведены диаграмма ER-типа и отношение, сформированное по правилу 1 на ее основе.

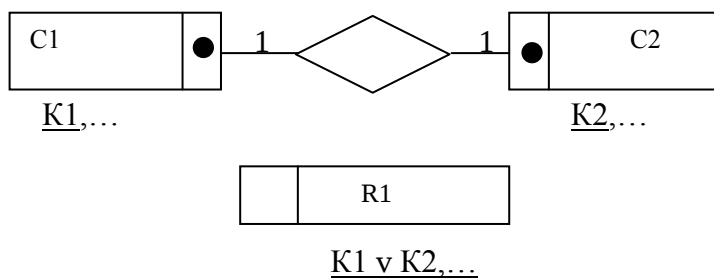


Рис. 4.8. Диаграмма и отношения для правила 1

На рисунке используются следующие обозначения:

C1, C2 – первая и вторая сущности соответственно;

K1, K2 – ключи первой и второй сущности соответственно;

R1 – отношение, сформированное на основе первой и второй сущностей;

$K1 \vee K2, \dots$ означает, что ключом сформированного отношения может быть либо K1, либо K2.

Примером использования первого правила, может быть, следующая ситуация. Имеются две сущности – Старосты и Группы; связь между ними один-к-одному, класс принадлежности обеих сущностей обязательный. В результате формируется одно отношение, содержащее информацию о группе в целом и ее старосте. Ключом отношения может быть ключ сущности Группы.

Правило 2. Если степень связи 1:1 и класс принадлежности одной сущности обязательный, а второй – необязательный, то для каждой из сущностей формируется по отношению с первичными ключами, являющимися ключами соответствующих сущностей. Далее к отношению, сущность которого имеет обязательный класс принадлежности, добавляется в качестве атрибута ключ сущности с необязательным классом принадлежности.

На рис. 4.9 приведены диаграмма ER-типа и отношения, сформированные по правилу 2 на ее основе.

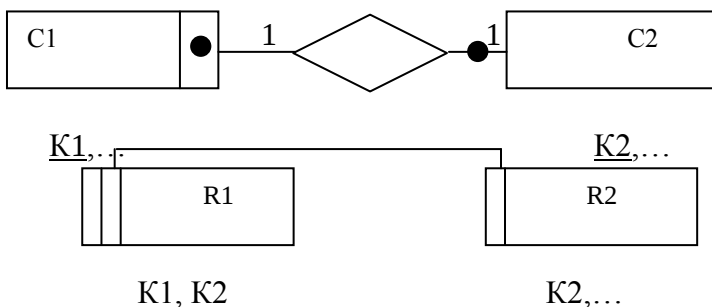


Рис. 4.9. Диаграмма и отношения для правила 2

Примером использования может быть такая ситуация. Имеются две сущности – Покупатели и Квартиры. Связь между ними – один-к-одному, класс принадлежности первой сущности – обязательный (если человек занесен в сущность Покупатели, значит он обязательно покупает квартиру), а второй – необязательный (не каждая квартира, принадлежащая сущности Квартиры, куплена). В результате формируется два отношения Покупатели (в которое добавляется электронный номер купленной квартиры) и Квартиры.

Правило 3. Если степень связи 1:1 и класс принадлежности обеих сущностей является необязательным, то необходимо создать три отношения. Два отношения соответствуют связываемым сущностям, ключи которых являются первичными ключами в этих отношениях. Третье отношение является связным между первыми двумя, поэтому его ключ объединяет ключевые атрибуты связываемых отношений.

На рис. 4.10 приведены диаграмма ER-типа и отношения, сформированные по правилу 3 на ее основе.

Примером использования третьего правила может быть следующая ситуация. Имеются две сущности Сотрудники фирмы и Товары и услуги. Рассматривается ситуация оформления товарного чека при покупке. Каждый чек может содержать один товар или услугу и оформляется одним продавцом. Класс при-

надлежности обеих сущностей необязательный (не каждый сотрудник оформляет чеки, необязательно каждый товар продается). В результате создается три отношения – Сотрудники, Товары и услуги, Товарный чек. Товарный чек содержит ключи двух сущностей (номер сотрудника, номер товара), а также может содержать дополнительные атрибуты – номер чека, дата продажи и количество. Номер чека может стать ключом третьего отношения.

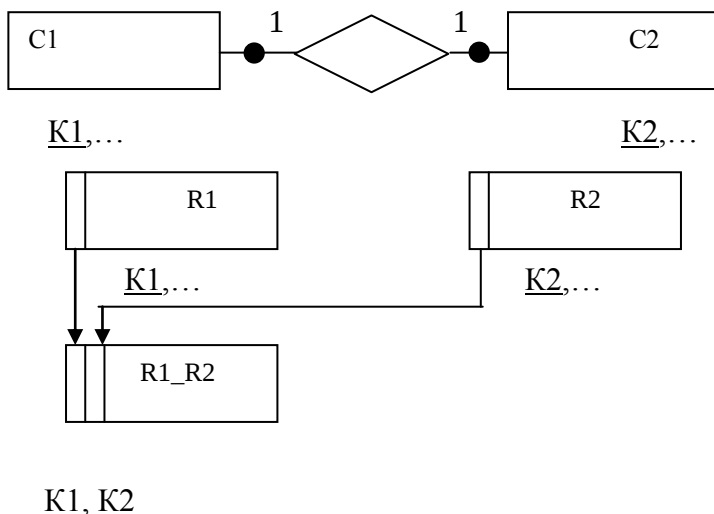


Рис. 4.10. Диаграмма и отношения для правила 3

Формирование отношений для связи 1:M.

Если две сущности C1 и C2 связаны как 1:M, сущность C1 будет называть односвязной (1-связной), а сущность C2 – многосвязной (М-связной). Определяющим фактором при формировании отношений, связанных этим видом связи, является класс принадлежности М-связной сущности. Так, если класс принадлежности М-связной сущности обязательный, то в результате применения правила получим два отношения, если не-

обязательный – три отношения. Класс принадлежности одно-связной сущности не влияет на результат.

Правило 4. Если степень связи между сущностями 1:M (или M:1) и класс принадлежности M-связной сущности обязательный, то достаточно формирования двух отношений (по одному на каждую из сущностей). При этом первичными ключами этих отношений являются ключи их сущностей. Кроме того, ключ 1-связной сущности добавляется как атрибут (внешний ключ) в отношение, соответствующее M-связной сущности.

На рис. 4.11 приведены диаграмма ER-типа и отношения, сформированные по правилу 4.

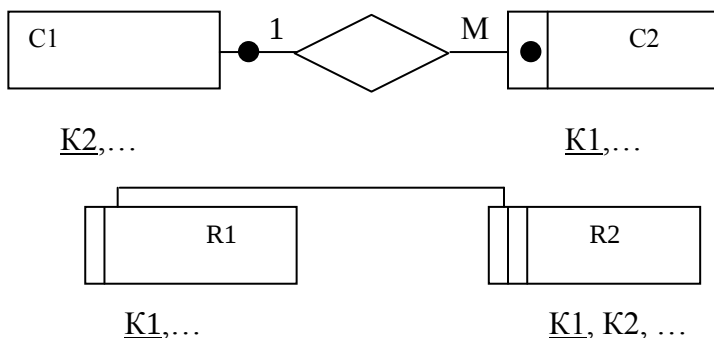


Рис. 4.11. Диаграмма и отношения для правила 4

Четвертое правило может проиллюстрировать следующая ситуация. Имеются две сущности – Факультеты и Специальности. Факультет может выпускать несколько специальностей и обязательно хотя бы одну. Специальности обязательно выпускаются на каком-то факультете и, как правило, только на одном. В результате формируются два отношения – Факультеты и Специальности, в отношение Специальности добавляется ключ (название факультета) из отношения Факультеты.

Правило 5. Если степень связи 1:M (M:1) и класс принадлежности M-связной сущности является необязательным, то необходимо формирование трех отношений (рис. 4.12).

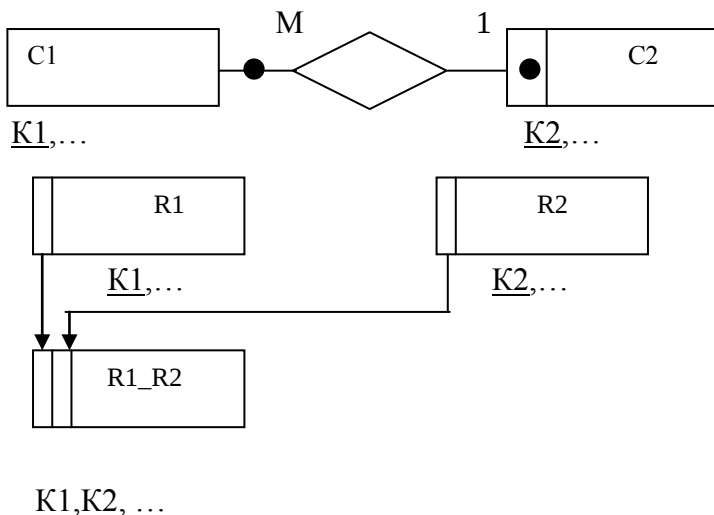


Рис. 4.12. Диаграмма и отношения для правила 5

На рис. 4.12 два отношения соответствуют связываемым сущностям, ключи которых являются первичными в этих отношениях. Третье отношение является связным между первыми двумя (его ключ объединяет ключевые атрибуты связываемых отношений).

Пятое правило может проиллюстрировать следующая ситуация. Имеются две сущности Стройматериалы и Склад. Один склад поставляет множество материалов, но не каждый материал может участвовать в поставке. Формируются три отношения – Склад, Стройматериалы, Поставка. Отношение Поставка содержит ключи исходных сущностей (номер склада, код материала) и дополнительные атрибуты – номер поставки, дата поставки, количество материала. Номер поставки может быть ключом третьего отношения.

Формирование отношений для связи М:М.

При наличии связи М:М между двумя сущностями необходимо три отношения независимо от класса принадлежности любой из сущностей. Использование одного или двух отноше-

ний в этом случае не избавляет от пустых полей или избыточного дублирования данных.

Правило 6. Если степень связи М:М, то независимо от класса принадлежности сущностей формируются три отношения. Два отношения соответствуют связываемым сущностям и их ключи являются первичными ключами этих отношений. Третье отношение является связным между первыми двумя, а его ключ объединяет ключевые атрибуты связываемых отношений.

На рис. 4.13 приведены диаграмма ER-типа и отношения, сформированные по правилу 6. На рисунке показан вариант с необязательным классом принадлежности обеих сущностей.

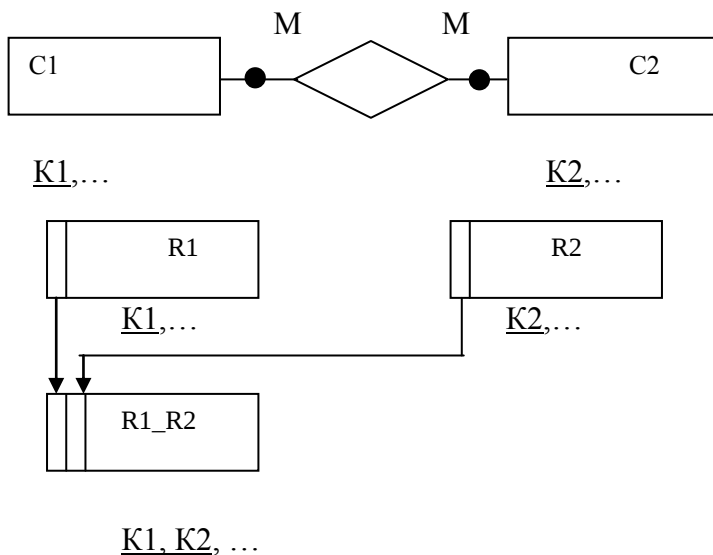


Рис. 4.13. Диаграмма и отношения для правила 6

Шестое правило может проиллюстрировать следующий пример. Имеются две сущности - Специальности и Дисциплины со связью многие-ко-многим (одна специальность изучает много дисциплин и одна дисциплина может изучаться несколькими

специальностями). В результате проектирования получатся три отношения – Специальности, Дисциплины, Учебный план (который будет содержать код специальности, номер дисциплины, семестр, количество часов лекций, лабораторных работ и т.д.).

4.5. Средства автоматизации проектирования

4.5.1. Основные определения

Для автоматизации проектирования и разработки информационных систем были созданы программно-технологические средства, получившие название CASE-средств и реализующие CASE-технологии создания и сопровождения информационных систем.

Термин CASE (Computer Aided Software Engineering) дословно переводится как разработка программного обеспечения с помощью компьютера. В настоящее время этот термин получил более широкий смысл, означающий автоматизацию разработки информационных систем.

CASE-средства представляют собой программные средства, поддерживающие процессы создания и/или сопровождения информационных систем. Основные операции, выполняемые данными средствами, - это анализ и формулировка требований, проектирование баз данных и приложений, генерация кода, тестирование, обеспечение качества, управление конфигурацией и проектом.

CASE-систему можно определить как набор CASE-средств, имеющих определенное функциональное предназначение и выполненных в рамках единого программного продукта.

CASE-технология обычно определяется как методология проектирования информационных систем плюс инструментальные средства, позволяющие наглядно моделировать предметную область, анализировать ее модель на всех этапах разработки и сопровождения информационной системы и разрабатывать приложения для пользователей.

CASE-индустрия объединяет сотни фирм и компаний различной ориентации. Практически все серьезные зарубежные программные проекты осуществляются с использованием CASE-средств, а общее число распространяемых пакетов превышает 500 наименований.

Основная цель CASE-систем и средств состоит в том, чтобы отделить проектирование программного обеспечения от его кодирования и последующих этапов разработки (тестирование, документирование и пр.), а также автоматизировать весь процесс создания программных систем, или инжиниринг (от англ. engineering – разработка).

В последнее время все чаще разработка программ начинается с некоторого предварительного варианта системы. В качестве такого варианта может выступать специально для этого разработанный прототип либо устаревшая и не удовлетворяющая новым требованиям система. В последнем случае для восстановления знаний о программной системе с целью последующего их использования применяют повторную разработку – реинжиниринг.

Повторная разработка сводится к построению исходной модели программной системы путем исследования ее программных кодов. Имея модель, можно ее усовершенствовать, а затем вновь перейти к разработке. Так часто и поступают при проектировании. Одним из наиболее известных принципов такого типа является принцип «возвратного проектирования» – Round Trip Engineering (RTE).

Современные CASE-системы не ограничиваются только разработкой, а чаще всего обеспечивают и повторную разработку. Это существенно ускоряет создание приложений и повышает их качество.

4.5.2. Модели жизненного цикла

Каждая CASE-система ориентирована на определенную модель жизненного цикла (ЖЦ) программного обеспечения (ПО) информационной системы (ИС).

Жизненный цикл ПО ИС представляет собой непрерывный процесс, начинающийся с момента принятия решения о создании ПО и заканчивающийся при завершении его эксплуатации.

Основным нормативным документом, регламентирующим ЖЦ ПО, является международный стандарт ISO/IEC 12207 (International Organization of Standardization – Международная организация по стандартизации / International Electrotechnical Commission – международная комиссия по электротехнике). В нем определяется структура ЖЦ, содержащая процессы, действия и задачи, которые должны быть выполнены при создании ПО.

Под **моделью ЖЦ ПО** понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач на протяжении ЖЦ. Наибольшее распространение получили следующие модели ЖЦ ПО: каскадная, с промежуточным контролем и спиральная.

Модели каскадная и с промежуточным контролем включают следующие этапы жизненного цикла ПО: анализ, проектирование, реализация, внедрение и сопровождение.

Каскадная модель предполагает строго последовательную реализацию перечисленных этапов жизненного цикла. Достоинствами такой модели являются: формирование на каждом этапе законченного комплекта документации, возможность планирования сроков завершения работ и соответствующих затрат. Недостатком модели является ее несоответствие реальному процессу создания ПО, который обычно не укладывается в жесткую схему и требует возврата к предыдущим этапам для уточнения и пересмотра принятых решений.

Модель с промежуточным контролем приближает жизненный цикл к реальному процессу создания и применения ПО. В отличие от каскадной модели, она допускает возврат с каждого этапа жизненного цикла на любой предыдущий этап для выполнения меж-этапной корректировки. При этом обеспечи-

вается большая надежность ПО, но вместе с тем увеличивается длительность периода разработки.

Спиральная модель жизненного цикла позволяет устранить недостатки предыдущих моделей. Основной упор в ней делается на начальные этапы: анализ и проектирование. На них реализуемость технических решений проверяется с помощью создания прототипов. При спиральной схеме разработки неполное завершение работ на очередном этапе позволяет переходить на следующий этап. Незавершенная работа может выполняться на следующем витке спирали. Тем самым обеспечивается возможность предъявить пользователям системы ее некоторый работоспособный вариант для уточнения требований.

4.5.3. Модели структурного проектирования

Структурный подход к анализу и проектированию информационной системы заключается в рассмотрении ее с общих позиций с последующей детализацией и представлением в виде иерархической структуры. На верхнем уровне иерархии обычно представляется функциональное описание системы.

При проведении структурного анализа и проектирования для повышения наглядности используется графическое представление функций информационной системы и отношений между данными.

Наиболее распространенными моделями и диаграммами графического представления являются следующие:

диаграммы сущность-связь или ER-диаграммы – Entity-Relationship Diagrams (ERD) служат для наглядного представления схем баз данных;

диаграммы потоков данных – Data Flow Diagrams (DFD) служат для иерархического описания модели системы;

метод структурного анализа и проектирования – Structured Analysis and Design Technique (SADT), служащий для построения функциональной модели объекта;

схемы описания иерархии вход-обработка-выход – Hierarchy plus Input-Processing-Output (HIPO) служат для описания реализуемых программой функций и циркулирующих внутри нее потоков данных;

диаграммы Варнье-Орра служат для описания иерархической структуры системы с выделением элементарных составных частей, выделением процессов и указанием потоков данных для каждого процесса.

Рассмотрим важные и часто используемые в CASE-средствах диаграммы и модели DFD и SADT.

Диаграммы потоков данных DFD лежат в основе методологии моделирования потоков данных, при котором модель системы строится как иерархия диаграмм потоков данных, описывающих процесс преобразования от ее входа до выдачи пользователю.

Диаграммы верхних уровней иерархии, или контекстные диаграммы, определяют основные процессы или подсистемы информационной системы с внешними входами и выходами. Их декомпозиция приводит к построению диаграмм более низкого уровня, которые в свою очередь также могут быть подвергнуты декомпозиции.

Основными компонентами диаграмм потоков данных являются:

- *внешние сущности* – источники или потребители информации, порождающие или принимающие информационные потоки (потоки данных);
- *системы/подсистемы*, преобразующие получаемую информацию и порождающие новые потоки;
- *процессы*, представляющие преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом;
- *накопители данных*, представляющие собой абстрактное устройство для хранения информации, которую можно поместить в накопитель и через некоторое время извлечь;

- *потоки данных*, определяющие информацию, передаваемую через некоторое соединение от источника к приемнику.

Рассмотрим типовой набор графических блоков, который обычно используют для обозначения компонентов DFD. В конкретных CASE-средствах и системах этот набор может иметь некоторые отличия.

Внешняя сущность обозначается прямоугольником с тенью.

Система и подсистема изображаются в форме прямоугольника с полями: номер, имя с определениями и дополнениями и имя проектировщика.

Процесс изображается в форме прямоугольника с полями: номер, имя процесса (содержит наименование процесса в виде предложения сделать что-то), физическая реализация (указывает, какое подразделение, программа или устройство выполняет процесс).

Накопитель данных изображается в форме прямоугольника без правой (или правой и левой) линии границы: идентификатор (буква D с числом) и имя (указывает на хранимые данные).

Поток данных изображается линией со стрелкой, показывающей направление потока, и именем, отражающим его содержание.

Примеры фрагментов диаграммы потоков данных с изображением перечисленных компонентов приведены на рис. 4.14.

Построение иерархии диаграмм потоков данных начинается с построения контекстной диаграммы. В случае простой информационной системы можно ограничиться одной контекстной диаграммой. Применительно к сложной информационной системе требуется построение иерархии контекстных диаграмм. При этом контекстная диаграмма верхнего уровня содержит набор подсистем, соединенных потоками данных.

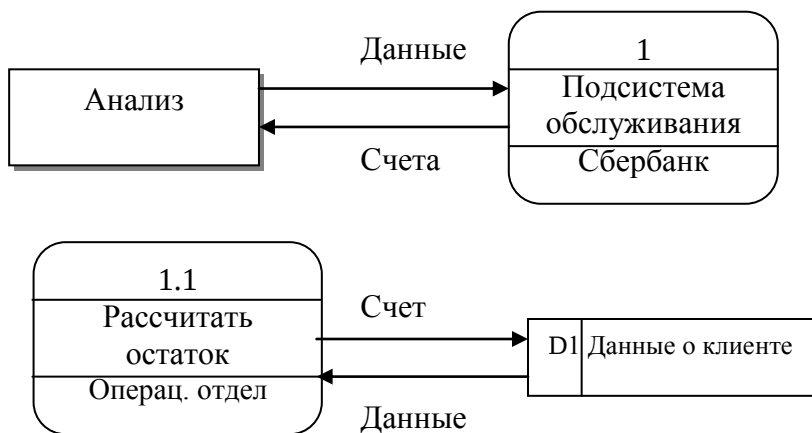


Рис. 4.14. Фрагменты диаграммы потоков данных

Методология функционального моделирования SADT служит для построения функциональной модели объекта какой-либо предметной области. Модель отображает функциональную структуру объекта, выполняемые им действия и связи между ними.

Функциональная модель информационной системы состоит из набора диаграмм, имеющих ссылки друг на друга, фрагментов текстов и глоссария. На диаграммах представляются функции ИС и взаимосвязи (интерфейсы) между ними в виде блоков и дуг. Место соединения дуги с блоком определяет тип интерфейса. Управляющая информация указывается сверху, обрабатываемая информация – с левой стороны блока, выводимая информация – с правой стороны, выполняющий операцию механизм (человек, программа или устройство) представляется дугой снизу блока (см. рис.4.15).

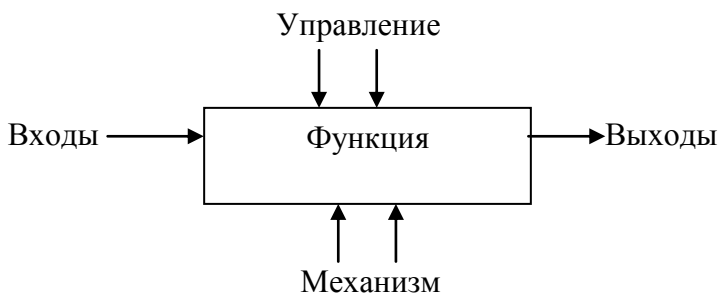


Рис. 4.15. Функциональный блок и дуги интерфейса

При использовании методологии SADT выполняется постепенное наращивание степени детализации в построении модели информационной системы. На рис. 4.16 показана декомпозиция исходного блока системы на три составляющих компонента. Каждый из блоков определяет подфункции исходной функции и, в свою очередь, может быть декомпозирован аналогичным образом для обеспечения большей детализации.

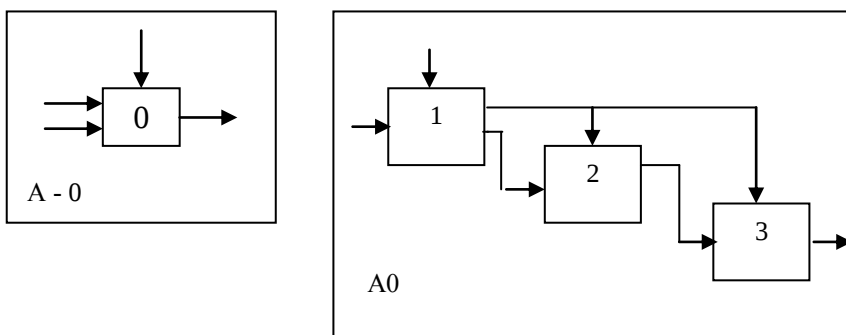


Рис. 4.16. Декомпозиция диаграмм

В общем случае функциональная модель ИС представляет собой серию диаграмм с документацией, декомпозирующих сложный объект на составные компоненты в виде блоков. Бло-

ки на диаграмме нумеруются. Для указания положения диаграммы или блока в иерархии диаграмм используются номера диаграмм. Например, обозначение А32 указывает на диаграмму, детализирующую блок 2 на диаграмме А3. В свою очередь, диаграмма А03 детализирует блок 3 на диаграмме А0.

На диаграммах функциональной модели SADT последовательность и время явно не указываются. Обратные связи, итерации, процессы и перекрывающиеся во времени функции можно отобразить с помощью дуг.

В методологии функционального моделирования важным является отображение возможных типов связей между функциями. Можно выделить следующие семь типов связей в порядке возрастания их значимости:

- *случайные связи*, означающие, что связь между функциями мала или отсутствует;
- *логические связи*, означающие, что данные и функции относятся к одному классу или набору элементов, но функциональных отношений между ними нет;
- *временные связи* представляют собой функции, связанные во времени, когда данные используются одновременно или функции включаются параллельно, а не последовательно;
- *процедурные связи* означают, что функции группируются вместе, так как выполняются в течение одной и той же части цикла или процесса;
- *коммуникационные связи* означают, что функции группируются вместе, так как используют одни и те же входные данные и/или порождают одни и те же выходные данные;
- *последовательные связи* служат для обозначения причинно-следственной зависимости – выходные данные одной функции являются входными данными другой функции;
- *функциональные связи* обозначают случай, когда все элементы функции влияют на выполнение только одной функции.

4.5.4. Объектно-ориентированные модели

Большинство моделей объектно-ориентированного проектирования близки по возможностям, но имеют отличия в основном в форме представления. Популярность объектно-ориентированных технологий привела к сближению большинства известных моделей. Многообразие моделей порождает трудности проектировщиков по выбору модели и обмену информацией при работе над разными проектами. В этой связи известные специалисты Г.Буч, Д.Рамбо, И. Джекобсон при поддержке фирмы Rational Software Corporation провели работу над унифицированной моделью и методом, получившим название UML (Unified Modeling Language – унифицированный язык моделирования).

Общая характеристика UML. UML представляет собой единый язык моделирования, предназначенный для спецификации, визуализации, конструирования и документирования материалов программных систем, а также для моделирования бизнеса и других непрограммных систем. В основу создания UML положены три наиболее распространенные модели:

- Booch, получившая название по фамилии автора Гради Буча (Grady Booch);
- OMT (Object Modeling Technique – метод моделирования объектов);
- OOSE (Object-Oriented Software Engineering – объектно-ориентированное проектирование программного обеспечения).

UML можно определить так же, как промышленный объектно-ориентированный стандарт моделирования. Он включает в себя в унифицированном виде лучшие методы визуального (графического) моделирования. В настоящее время имеется целый ряд инструментальных средств, производители которых заявляют о поддержке UML, среди них можно выделить: Rational Rose, Select Enterprise, Platinum, Visual Modeler.

Типы диаграмм UML. Создаваемый с помощью UML проект информационной системы может включать в себя сле-

дующие 8 видов диаграмм: прецедентов использования; классов; состояний; активности; следования; сотрудничества; компонентов; размещения.

Диаграммы состояний, активности, следования и сотрудничества образуют набор диаграмм, служащих для описания поведения разрабатываемой информационной системы. Причем последние две обеспечивают описание взаимодействия объектов информационной системы. Диаграммы компонентов и размещения описывают физическую реализацию информационной системы.

Диаграммы **прецедентов использования** описывают функциональность ИС, видимую пользователями системы. Каждая функциональность изображается в виде прецедентов использования. Прецедент – это типичное взаимодействие пользователя с системой, которое выполняет следующее: описывает видимую пользователем функцию; представляет различные уровни детализации; обеспечивает достижение конкретной цели.

Прецедент изображается как овал, связанный с типичными пользователями, называемыми «актерами» (actors). Актером является любая сущность, взаимодействующая с системой извне, например человек, оборудование, другая система. Прецедент описывает, что система предоставляет актеру – определяет набор транзакций, выполняемый актером при диалоге с информационной системой. На диаграмме изображается один актер, но пользователей, выступающих в роли актера, может быть много. Диаграмма прецедентов использования имеет высокий уровень абстракции и позволяет определить функциональные требования к ИС.

Диаграммы **классов** описывают статическую структуру классов. В состав диаграмм классов входят следующие элементы: классы, объекты и отношения между ними. Класс представляется прямоугольником, включающим три раздела: имя класса, атрибуты и операции. Аналогичное обозначение применяется и

для объектов, но к имени класса добавляется имя объекта и вся надпись подчеркивается.

Диаграммы **состояний** описывают поведение объекта во времени, моделируют все возможные изменения в состоянии объекта, вызванные внешними воздействиями со стороны других объектов или извне. Этот тип диаграмм описывает изменение состояния одного класса или объекта. Каждое состояние объекта представляется в виде прямоугольника с закругленными углами, содержащего имя состояния и, возможно, значение атрибутов объекта в данный момент времени. Переход осуществляется при наступлении некоторого события (например, получения объектом сообщения или приема сигнала) и изображается в виде стрелки, соединяющей два соседних сообщения. Имя события указывается на переходе. На переходе могут указываться также действия, производимые объектом в ответ на внешние события.

Диаграммы **активности** представляют частный случай диаграмм состояний. Каждое состояние есть выполнение некоторой операции, и переход в следующее состояние происходит при завершении операции. Для диаграмм активности используются аналогичные диаграммам состояний обозначения, но на переходах отсутствует сигнатура события и добавлен символ «синхронизации» переходов для реализации параллельных алгоритмов. Диаграммы активности используются в основном для описания операций классов.

Диаграмма **следования** определяет временную последовательность (динамику взаимодействия) передаваемых сообщений, порядок, вид и имя сообщения. На диаграмме изображаются объекты, непосредственно участвующие во взаимодействии.

Диаграммы **сотрудничества** описывают взаимодействие объектов системы, выполняемого ими для получения некоторого результата. Под получением результата подразумевается выполнение законченного действия.

Диаграмма сотрудничества изображает объекты, участвующие во взаимодействии, в виде прямоугольников, содержа-

щих имя объекта, его класс и, возможно, значение атрибутов. Ассоциации между объектами изображаются в виде соединительных линий. Возможно указание имени ассоциации и ролей объектов в данной ассоциации. Динамические связи (потoki сообщений) представляются в виде соединительных линий между объектами, сверху которых располагается стрелка с указанием направления и имени сообщения.

Диаграмма **компонентов** служит для определения архитектуры разрабатываемой системы путем установления зависимости между программными компонентами: исходным, бинарным и/или исполняемым кодом.

Диаграммы **размещения** используются для задания конфигурации компонентов, процессов и объектов, действующих в системе на этапе выполнения. Кроме того, они показывают физическую зависимость аппаратных устройств, участвующих в реализации системы, и соединений между ними – маршрутов передачи информации.

Примеры диаграмм UML. В качестве предметной области используем описание работы библиотеки, которая получает запросы от клиентов на различные издания и регистрирует информацию об их возвращении в фонды библиотеки.

Пример диаграммы прецедентов использования приведен на рис. 4.17.

На диаграмме приведен ряд выделенных при анализе и реализуемых информационной системой функций: администрирование пользователей; учет книг; составление отчетов и поиск издания.

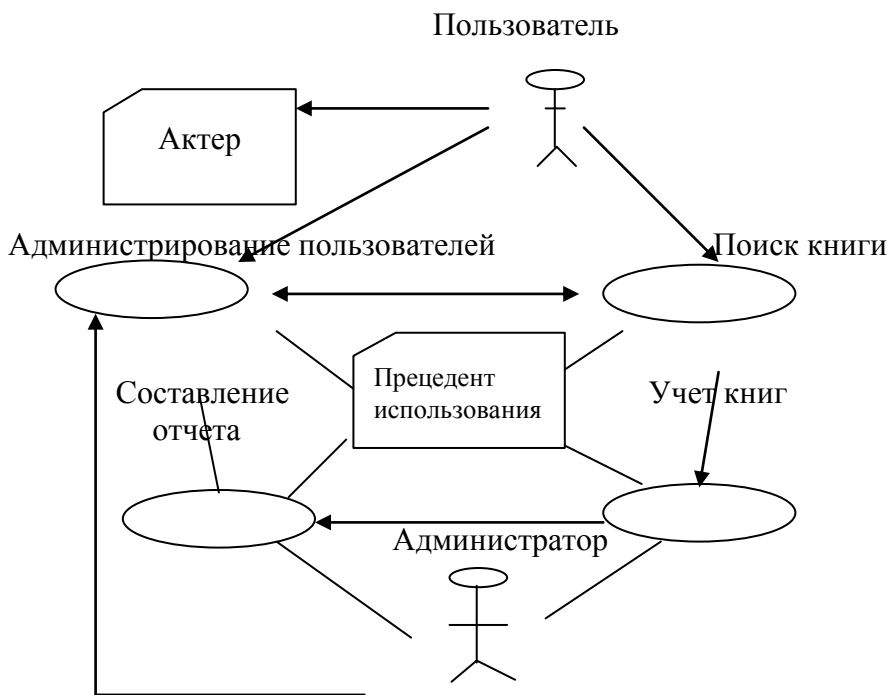


Рис. 4.17. Диаграмма прецедентов использования

4.5.5. Классификация CASE-средств

При классификации CASE-средств используют следующие признаки:

- ориентацию на этапы жизненного цикла;
- функциональную полноту;
- тип используемой модели;
- степень независимости от СУБД;
- допустимые платформы.

Рассмотрим классификацию CASE-средств по наиболее часто используемым признакам.

По ориентации на этапы жизненного цикла выделяют следующие основные типы CASE-средств:

- средства анализа, предназначенные для построения и анализа моделей предметной области, например: Design/IDEF (Meta Software), BPwin (Logic Works);

- средства анализа и проектирования, обеспечивающие создание проектных спецификаций, например: Vantage Team Builder (Cayenne), Silverrun (Silverrun Technologies), PRO-IV (McDonnell Douglas), CASE Аналитик (МакроПроджект);

- средства проектирования баз данных, обеспечивающие моделирование данных и разработку схем баз данных для основных СУБД, например: ERwin (Logic Works), S-Designor (SPD), DataBase Designer (ORACLE);

- средства разработки приложений, например: Uniface (Compuware), JAM (JYACC), PowerBuilder (Sybase), Developer/2000 (ORACLE), New Era (Informix), SQL Windows (Centura), Delphi (Borland).

По **функциональной полноте** CASE-системы и средства можно условно разделить на следующие типы:

- системы, предназначенные для решения частных задач на одном или нескольких этапах жизненного цикла, например, ERwin, S-Designor, CASE.Аналитик, Silerrun;

- интегрированные системы, поддерживающие весь жизненный цикл ИС и связанные с общим репозиторием, например, система Vantage Team Builder, система Designer/2000 с системой разработки приложений Developer/2000

По **типу используемых моделей** CASE-системы условно можно разделить на три основные разновидности: структурные, объектно-ориентированные и комбинированные.

Исторически первые *структурные* CASE-системы основаны на методах структурного и модульного программирования, структурного анализа и синтеза, например, система Vantage Team Builder.

Объектно-ориентированные методы и CASE-системы получили массовое использование с начала 90-х годов. Они позволяют сократить сроки разработки, а также повысить надежность и эффективность функционирования ИС. Примерами

объектно-ориентированных CASE-систем являются Rational Rose и Object Team.

Комбинированные инструментальные средства поддерживают одновременно структурные и объектно-ориентированные методы, например: Designer/2000.

По **степени независимости от СУБД** CASE-системы можно разделить на две группы: независимые системы, встроенные в СУБД.

Независимые CASE-системы поставляются в виде автономных систем, не входящих в состав конкретной СУБД. Обычно они поддерживают несколько форматов баз данных через интерфейс ODBC. К числу независимых CASE-систем относятся S-Designer, ERwin, Silverrun.

Встроенные CASE-системы обычно поддерживают главным образом формат баз данных СУБД, в состав которой они входят. При этом возможна поддержка и других форматов баз данных. Примером встроенной системы является Designer/2000, входящая в состав СУБД ORACLE.

5. ФИЗИЧЕСКИЕ МОДЕЛИ БАЗ ДАННЫХ

5.1. Файловые структуры, используемые в базах данных

В каждой СУБД по-разному организованы хранение и доступ к данным, однако существуют некоторые файловые структуры, которые имеют общепринятые способы организации и широко применяются практически во всех СУБД.

В системах баз данных файлы и файловые структуры, которые используются для хранения информации во внешней памяти, можно классифицировать следующим образом (рис. 5.1).

С точки зрения пользователя, файлом называется поименованная линейная последовательность записей, расположенных на внешних носителях. Так как файл – это линейная последовательность записей, то всегда в файле можно определить текущую запись, предшествующую ей и следующую за ней. Всегда существует понятие первой и последней записи файла.

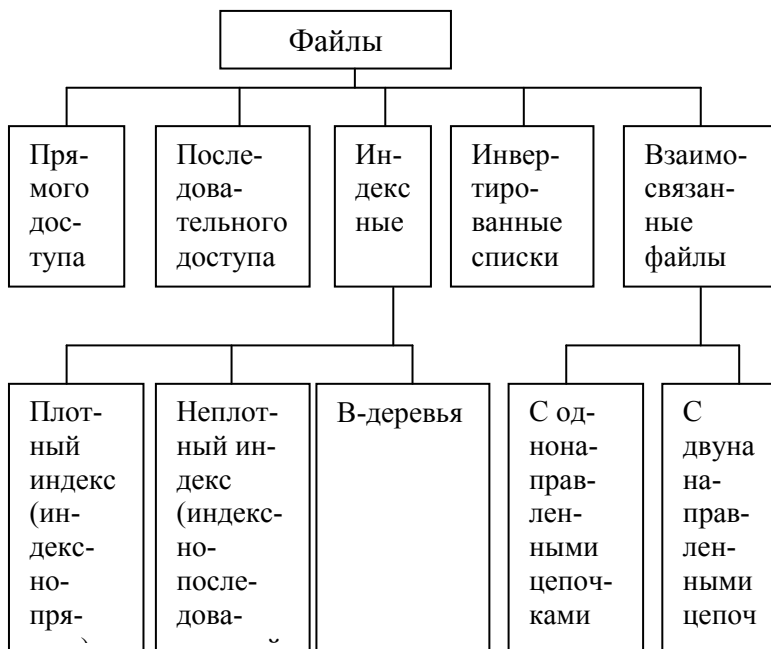


Рис. 5.1. Классификация файлов, используемых в системах баз данных

В соответствии с методами управления доступом различают устройства внешней памяти с произвольной адресацией (магнитные и оптические диски) и устройства с последовательной адресацией (магнитофоны, стримеры).

На устройствах с произвольной адресацией теоретически возможна установка головок чтения-записи в произвольное место мгновенно. Практически существует время позиционирования головки, которое весьма мало по сравнению со временем считывания-записи.

В устройствах с последовательным доступом для получения доступа к некоторому элементу требуется «перемотать (пройти)» все предшествующие ему элементы информации. На устройствах с последовательным доступом вся память рассмат-

ривается как линейная последовательность информационных элементов.

Файлы с постоянной длиной записи, расположенные на устройствах прямого доступа (УПД), являются *файлами прямого доступа*.

В этих файлах физический адрес расположения нужной записи может быть вычислен по номеру записи.

Каждая файловая система СУФ – система управления файлами поддерживает некоторую иерархическую файловую структуру, включающую чаще всего неограниченное количество уровней иерархии в представлении внешней памяти (рис. 5.2).

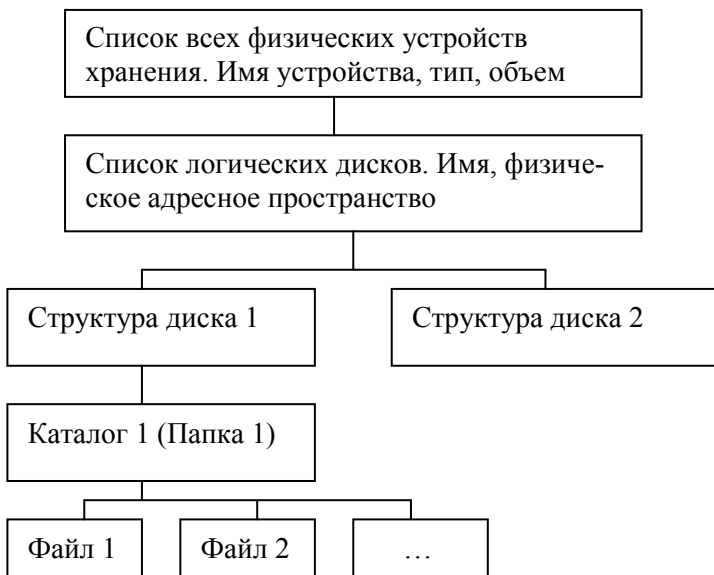


Рис. 5.2. Иерархическая организация файловой структуры хранения

На устройствах последовательного доступа могут быть организованы файлы только последовательного доступа.

Файлы с переменной длиной записи всегда являются файлами последовательного доступа. Они могут быть организованы двумя способами:

конец записи отмечается специальным маркером;
в начале каждой записи записывается ее длина.

Файлы с прямым доступом обеспечивают наиболее быстрый способ доступа. Однако доступ по номеру записи в базах данных весьма неэффективен. Чаще всего в базах данных необходим поиск по первичному или возможному ключам, иногда необходима выборка по внешним ключам, но во всех этих случаях мы знаем значение ключа, но не знаем номера записи, который соответствует этому ключу.

5.2. Хешированные файлы

При организации файлов прямого доступа возможно построение функции, которая по значению ключа однозначно вычисляет адрес (номер записи файла). Функция при этом должна быть линейной, чтобы обеспечить однозначное соответствие между ключом и номером записи.

Однако далеко не всегда удастся построить взаимно-однозначное соответствие между значениями ключа и номерами записей. Часто бывает, что значения ключей разбросаны по нескольким диапазонам. В этом случае не удастся построить взаимно-однозначную функцию, либо эта функция будет иметь множество незадействованных значений, которые соответствуют недопустимым значениям ключа. В подобных случаях применяют различные методы хеширования (рандомизации) и создают специальные хеш-функции.

Метод хеширования предполагает, что по значению ключа вычисляется некоторая хеш-функция и полученное значение берется в качестве адреса начала поиска. Допускается, что нескольким разным ключам может соответствовать одно значение хеш-функции (то есть один адрес). Подобные ситуации называются коллизиями. Значения ключей, которые имеют одно и то

же значение хеш-функции, называются синонимами. Поэтому при использовании хеширования как метода доступа необходимо принять два решения:

- выбрать хеш-функцию;
- выбрать метод разрешения коллизий.

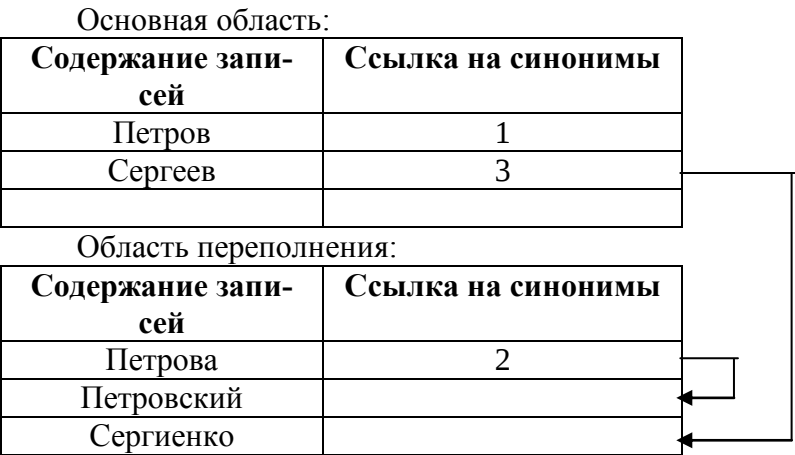
Существует множество различных стратегий разрешения коллизий. Однако наиболее распространенными являются: стратегия с областью переполнения, стратегия свободного замещения.

5.2.1. Стратегия разрешения коллизий с областью переполнения

При выборе этой стратегии область хранения разбивается на две части:

- основную область;
- область переполнения.

Для каждой новой записи вычисляется значение хеш-функции, которое определяет адрес ее расположения, и запись заносится в основную область в соответствии с полученным значением хеш-функции.



Если вновь заносимая запись имеет значение функции хеширования такое же, которое использовала другая запись, уже имеющаяся в БД, то новая запись заносится в область переполнения на первое свободное место, а в записи-синониме, которая находится в основной области, делается ссылка на адрес вновь размещенной записи в области переполнения. Если же уже существует ссылка в записи-синониме, которая расположена в основной области, то тогда новая запись получает дополнительную информацию в виде ссылки и уже в таком виде заносится в область переполнения.

При этом цепочка синонимов не разрывается, но мы не просматриваем ее до конца, чтобы расположить новую запись в конце цепочки синонимов, а располагаем всегда новую запись на второе место в цепочке синонимов, что существенно сокращает время размещения новой записи. Запись в основной области теперь ссылается на номер расположения новой записи в области переполнения, а новая запись в области переполнения получает ссылку, которая была у записи в основной области. При таком алгоритме время размещения любой новой записи составляет не более двух обращений к диску, с учетом того, что номер первой свободной записи в области переполнения хранится в виде системной переменной.

При **поиске** записи также сначала вычисляется значение ее хеш-функции и считывается первая запись в цепочке синонимов, которая расположена в основной записи. Если искомая запись не соответствует первой в цепочке синонимов, то далее поиск происходит перемещением по цепочке синонимов, пока не будет обнаружена требуемая запись. Скорость поиска зависит от длины цепочки синонимов, поэтому качество хеш-функции определяется максимальной длиной цепочки синонимов. Хорошим результатом может считаться наличие не более 10 синонимов в цепочке.

При **удалении** произвольной записи сначала определяется ее место расположения. Если удаляемой является первая запись в цепочке синонимов, то после удаления на ее место в основной

области заносится вторая (следующая) запись в цепочке синонимов, при этом все указатели (ссылки на синонимы) сохраняются.

Если же удаляемая запись находится в середине цепочки синонимов, то необходимо провести корректировку указателей: в записи, предшествующей удаляемой в цепочке, ставится указатель из удаляемой записи. Если это последняя запись в цепочке, то все равно механизм изменения указателей такой же, то есть в предшествующую запись заносится признак отсутствия следующей записи в цепочке, который ранее хранился в последней записи.

5.2.2. Организация стратегии свободного замещения

При этой стратегии пространство не разделяется на области, но для каждой записи добавляется два указателя: указатель на предыдущую запись в цепочке синонимов и указатель на следующую запись в цепочке синонимов. Отсутствие соответствующей ссылки обозначается специальным символом, например нулем. Для каждой новой записи вычисляется значение хеш-функции, и если данный адрес свободен, то запись попадает на заданное место и становится первой в цепочке символов. Если адрес, соответствующий полученному значению хеш-функции, занят, то по наличию ссылок определяется, является ли запись, расположенная по указанному адресу, первой в цепочке синонимов. Если да, то новая запись располагается на первом свободном месте и для нее устанавливаются соответствующие ссылки: она становится второй в цепочке синонимов, на нее ссылается первая запись, а она ссылается на следующую, если таковая есть.

Если запись, которая занимает требуемое место, не является первой записью в цепочке синонимов, значит, она занимает данное место «незаконно» и при появлении «законного владельца» должна быть «выселена», то есть перемещена на новое место. Механизм перемещения аналогичен занесению новой

записи, которая уже имеет синоним, занесенный в файл. Для этой записи ищется первое свободное место и корректируются соответствующие ссылки: в записи, которая является предыдущей в цепочке синонимов для перемещаемой записи, заносится указатель на новое место перемещаемой записи, указатели же в самой перемещаемой записи остаются прежние.

После перемещения «незаконной» записи вновь вносимая запись занимает свое законное место и становится первой записью в новой цепочке синонимов.

Механизмы удаления записей во многом аналогичны механизмам удаления в стратегии с областью переполнения.

Если удаляемая запись является первой записью в цепочке синонимов, то после удаления на ее место перемещается следующая (вторая) запись из цепочки синонимов и проводится соответствующая корректировка указателя третьей записи в цепочке синонимов, если таковая существует.

Если же удаляется запись, которая находится в середине цепочки синонимов, то производится только корректировка указателей. В предшествующей записи указатель на удаляемую запись заменяется указателем на следующую за удаляемой запись, а в записи, следующей за удаляемой, указатель на предыдущую запись заменяется на указатель на запись, предшествующую удаляемой.

5.3. Индексные файлы

Несмотря на высокую эффективность хеш-адресации в файловых структурах далеко не всегда удается найти соответствующую функцию, поэтому при организации доступа по первичному ключу широко используются индексные файлы.

Индексные файлы можно представить как файлы, состоящие из двух частей. Необязательно эти две части находятся в одном файле. В большинстве случаев индексная область образует отдельный индексный файл.

Для описания механизма индексации, используемого для ускорения доступа к данным, будем рассматривать основную и индексную область совместно. Предполагается, что сначала идет индексная область, которая занимает некоторое целое число блоков, а затем идет основная область, в которой последовательно расположены все записи файла.

В зависимости от организации индексной и основной областей различают два типа файлов: с плотным индексом и с неплотным индексом. Эти файлы имеют еще дополнительные названия, которые связаны с методами доступа к произвольной записи.

Файлы с плотным индексом называются также индексно-прямыми файлами, а файлы с неплотным индексом называются также индексно-последовательными файлами.

5.3.1. Файлы с плотным индексом, или индексно-прямые файлы

В этих файлах основная область содержит последовательность записей одинаковой длины, расположенных в произвольном порядке, а индексные записи состоят из двух полей: значение ключа, номер записи. Значение ключа – это значение первичного ключа, а номер записи – это порядковый номер записи в основной области, которая имеет данное значение первичного ключа.

Если индексные файлы строятся для первичных ключей, однозначно определяющих запись, то в них не может быть двух записей, имеющих одинаковые значения первичного ключа. В индексных файлах с плотным индексом для каждой записи в основной области существует одна запись из индексной области. Все записи в индексной области упорядочены по значению ключа, поэтому можно применить более эффективные способы поиска в упорядоченном пространстве.

Длина доступа к произвольной записи оценивается не в абсолютных значениях, а в количестве обращений к устройству

внешней памяти, которым обычно является диск. Обращение к диску является наиболее длительной операцией по сравнению со всеми обработками в оперативной памяти.

Наиболее эффективным алгоритмом поиска на упорядоченном массиве является логарифмический или бинарный поиск. При этом все пространство поиска разбивается пополам, и так как оно строго упорядочено, то определяется сначала, не является ли элемент искомым, а если нет, то в какой половине его надо искать. На следующем шаге выбранную половину вновь делят пополам и проводят аналогичные сравнения пока не обнаружат искомый элемент. Максимальное количество шагов поиска определяется двоичным логарифмом от общего числа элементов в искомом пространстве поиска:

$$T_n = \log_2 N,$$

где N – число элементов.

Однако в нашем случае является существенным только число обращений к диску при поиске записи по заданному значению первичного ключа. Поиск происходит в индексной области, где применяется двоичный алгоритм поиска индексной записи, а потом путем прямой адресации мы обращаемся к основной области уже по конкретному номеру записи. Для того, чтобы оценить максимальное время доступа, нам необходимо определить количество обращений к диску для поиска произвольной записи.

На диске записи файлов хранятся в блоках. Размер блока определяется физическими особенностями дискового контроллера и операционной системой. В одном блоке может размещаться несколько записей. Поэтому надо определить количество индексных блоков, которое потребуется для размещения всех индексных записей, а потому максимальное число обращений к диску будет равно двоичному логарифму от заданного числа блоков плюс единица. После поиска номера записи в индексной области необходимо еще обратиться к основной области файла (отсюда появляется 1). Формула вычисления количества обращений к диску следующая:

$$T_n = \log_2 N_{\text{бл.инд.}} + 1.$$

Рассмотрим пример, позволяющий сравнить количество обращений к диску при последовательном просмотре и при организации плотного индекса.

Пусть имеются следующие исходные данные:

длина записи файла (LF) – 128 байт;

длина первичного ключа (LK) – 12 байт;

количество записей в файле (KZ) – 100 000;

размер блока (LB) – 1024 байта.

Рассчитаем размер индексной записи. Для представления целого числа в пределах 100 000 потребуется 3 байта. Если считать, что допустима только четная адресация, то надо отвести 4 байта для хранения номера записи, тогда длина индексной записи будет равна сумме размера ключа и ссылки на номер записи, то есть:

$$LI = LK + 4 = 12 + 4 = 16 \text{ байт.}$$

Определим количество индексных блоков, которое требуется для обеспечения ссылок на заданное количество записей. Для этого сначала определим, сколько индексных записей может храниться в одном блоке:

$KIZB = LB / LI = 1024 / 16 = 64$ индексных записи в одном блоке.

Теперь определим необходимое количество индексных блоков:

$$KIB = KZ / KIZB = 100\,000 / 64 = 1563 \text{ блока.}$$

Результат округлен в большую сторону, потому что пространство выделяется целыми блоками, и последний блок будет заполнен не полностью.

Максимальное количество обращений к диску при поиске произвольной записи:

$T_{\text{поиска}} = \log_2 KIB + 1 = \log_2 1563 + 1 = 11 + 1 = 12$ обращений к диску.

Логарифм в вычислениях также округлен.

Следовательно, для поиска произвольной записи по первичному ключу при организации плотного индекса потребуется не более 12 обращений к диску.

При отсутствии индексного пространства для поиска нужной записи необходимо просмотреть все блоки, в которых хранится файл, временем просмотра записей внутри блока пренебрегаем, так как этот процесс происходит в оперативной памяти.

Количество блоков, которое необходимо для хранения 100 000 записей, определяют по следующей формуле:

$$KBO = KZ / (LB / LZ) = 100\,000 / (1024/128) = 12500 \text{ бло-}$$

ков.

Выигрыш при использовании файла с плотным индексом очевиден (12 или 12500 обращений к диску).

Рассмотрим, как осуществляются операции добавления и удаления новых записей.

При операции добавления осуществляется запись в конец основной области. В индексной области необходимо произвести занесение информации в конкретное место, чтобы не нарушать упорядоченности. Поэтому вся индексная область файла разбивается на блоки и при начальном заполнении в каждом блоке остается свободная область (область расширения) (рис 5.3).

После определения блока, в который должен быть занесен индекс, этот блок копируется в оперативную память, там он модифицируется путем вставки в нужное место новой записи и измененный записывается обратно на диск.

Определим максимальное количество обращений к диску, которое требуется при добавлении записи, - это количество обращений, необходимое для поиска записи плюс одно обращение для занесения измененного индексного блока и плюс одно обращение для занесения записи в основную область.

$$T_{\text{добавления}} = \log_2 N + 1 + 1 + 1.$$

		Ссылка на номер записи	
Ключ			
Блок 1	01-30-01	5	Индексная область
	02-40-02	2	
	05-40-00	4	
	Свободная область		
Блок 2	06-40-00	1	
	06-50-01	6	
	07-35-00	3	
	Свободная область		
1	06-40-00	Васильев	Основная область
2	02-40-02	Лукина	
3	07-35-00	Сидоров	
4	05-40-00	Попова	
5	01-30-01	Архипова	
6	06-50-01	Павлов	

Рис. 5.3. Пример организации файла с плотным индексом

В процессе добавления новых записей область расширения постоянно уменьшается. Когда исчезнет свободная область, возникает переполнение индексной области. В этом случае возможны два решения: либо перестроить заново индексную область, либо организовать область переполнения для индексной области, в которой будут храниться не поместившиеся в основную область записи. Однако первый способ потребует дополнительного времени на перестройку индексной области, а второй увеличит время на доступ к произвольной записи и потребует организации дополнительных ссылок в блоках на область переполнения. Поэтому при проектировании физической базы данных важно заранее как можно точнее определить объемы хранимой информации, спрогнозировать ее рост и предусмотреть соответствующее расширение области хранения.

При удалении записи возникает следующая последовательность действий: запись в основной области помечается как

удаленная (отсутствующая), в индексной области соответствующий индекс уничтожается физически, то есть записи, следующие за удаленной записью, перемещаются на ее место и блок, в котором хранился данный индекс, заново записывается на диск. При этом количество обращений к диску для этой операции такое же, как и при добавлении новой записи.

5.3.2. Файлы с неплотным индексом, или индексно-последовательные файлы

Неплотный индекс строится для упорядоченных файлов. Если основной файл хранить в упорядоченном виде и применить к нему алгоритм двоичного поиска для доступа к произвольной записи, то время доступа будет существенно меньше. Для нашего примера составит:

$$T = \log_2 KBO = \log_2 12\,500 = 14 \text{ обращений к диску.}$$

Это существенно меньше, чем 12 500 обращений при произвольном хранении записей файла. Однако и поддержание основного файла в упорядоченном виде также операция сложная.

Неплотный индекс основывается на принципе внутреннего упорядочивания. Структура записи индекса для таких файлов имеет следующий вид: значение ключа первой записи блока; номер блока с этой записью.

В индексной области ищут нужный блок по заданному значению первичного ключа. Так как все записи упорядочены, то значение первой записи блока позволяет быстро определить, в каком блоке находится искомая запись. Все остальные действия происходят в основной области. На рис. 5.4 представлен пример заполнения основной и индексной областей, если первичным ключом являются целые числа.

Время сортировки больших файлов весьма значительно, но поскольку файлы поддерживаются сортированными с момента их создания, накладные расходы в процессе добавления новой информации будут гораздо меньше.

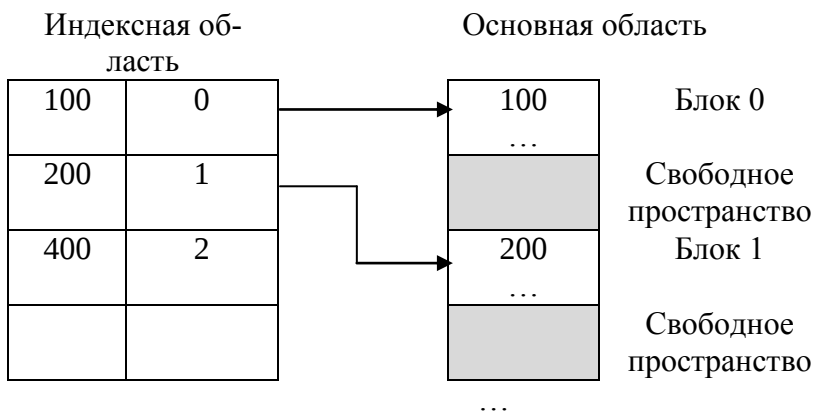


Рис. 5.4. Пример заполнения индексной и основной области при организации неплотного индекса

Оценим время доступа к произвольной записи для файлов с неплотным индексом. Алгоритм решения задачи аналогичен.

Сначала определим размер индексной записи. Если ранее ссылка рассчитывалась исходя из того, что требовалось ссылаться на 100 000 записей, то теперь нам требуется ссылаться всего на 12 500 блоков, поэтому для ссылки достаточно двух байт. Тогда длина индексной записи будет равна:

$$LI = LK + 2 = 14 + 2 = 14 \text{ байт.}$$

Тогда количество индексных записей в одном блоке будет равно:

$$KIZB = LB / LI = 1024 / 14 = 73 \text{ индексные записи в одном блоке.}$$

Определим количество индексных блоков, которое необходимо для хранения требуемых индексных записей:

$$KIB = KBO / KIZB = 12\,500 / 73 = 172 \text{ блока.}$$

Тогда время доступа по прежней формуле будет определяться:

$$T_{\text{поиска}} = \log_2 KIB + 1 = \log_2 172 + 1 = 8 + 1 = 9 \text{ обращений к диску.}$$

Таким образом, при переходе к неплотному индексу время доступа уменьшилось практически в полтора раза. Поэтому

можно признать, что организация неплотного индекса дает выигрыш в скорости доступа.

Рассмотрим процедуры добавления и удаления новой записи при подобном индексе.

Механизм включения новой записи принципиально отличен от ранее рассмотренного. Новая запись должна заноситься сразу в требуемый блок на требуемое место, которое определяется заданным принципом упорядоченности на множестве значений первичного ключа. Поэтому сначала ищут требуемый блок основной памяти, в который надо поместить новую запись, а потом этот блок считывается, затем в оперативной памяти корректируется содержимое блока, и он снова записывается на диск на старое место. Здесь, так же, как и в первом случае, должен быть задан процент первоначального заполнения блоков, но только применительно к основной области. В MS SQL server этот процент называется Full-factor и используется при формировании кластеризованных индексов. Кластеризованными называются индексы, в которых исходные записи физически упорядочены по значениям первичного ключа. При внесении новой записи индексная область не корректируется.

Количество обращений к диску при добавлении новой записи равно количеству обращений, необходимых для поиска соответствующего блока плюс одно обращение, которое требуется для занесения измененного блока на старое место.

$T_{\text{добавления}} = \log_2 N + 1 + 1$ обращений.

Уничтожение записи происходит путем ее физического удаления из основной области, при этом индексная область обычно не корректируется, даже если удаляется первая запись блока. Поэтому количество обращений к диску при удалении записи такое же, как и при добавлении новой записи.

5.3.3. Организация индексов в виде B-tree (B-деревьев)

Построение B-деревьев связано с построением индекса над уже построенным индексом. Если построен неплотный индекс, то сама индексная область может быть рассмотрена как

основной файл, над которым надо построить неплотный индекс, а потом снова над новым индексом можно построить следующий и так до того момента, пока не останется всего один индексный блок.

В результате получается некоторое дерево, каждый родительский блок которого связан с одинаковым количеством подчиненных блоков, число которых равно числу индексных записей, размещенных в одном блоке. Количество обращений к диску при этом для поиска любой записи одинаково и равно количеству уровней в построенном дереве. Такие деревья называются сбалансированными именно потому, что путь от корня до любого листа в этом дереве одинаков.

Построим подобное дерево для нашего примера и рассчитаем для него количество уровней и, соответственно, количество обращений к диску.

На первом уровне число блоков равно числу блоков основной области (12 500 блоков). Второй уровень образуется из неплотного индекса, число блоков индексной области в этом случае равно 172 блокам. Теперь над вторым уровнем снова построим неплотный индекс. Длина индексной записи по-прежнему будет равна 14 байтам. Количество индексных записей в одном блоке равно 73. Определим, сколько блоков необходимо для хранения ссылок на 172 блока:

$$KIB_3 = KIB_2 / KIZB = 172 / 73 = 3 \text{ блока.}$$

Над третьим уровнем строим новый, и на нем будет всего один блок, в котором будет всего три записи. Число уровней в построенном дереве равно четырем, и соответственно количество обращений к диску для доступа к произвольной записи равно четырем (рис. 5.5). Это не максимально возможное число обращений, а всегда одно и то же, одинаковое для доступа к любой записи.

$$T_d = R_{\text{уровней}} = 4.$$

Механизм добавления и удаления записи при организации индекса в виде В-дерева аналогичен механизму, применяемому в случае с неплотным индексом.

В случае плотного индекса после определения местонахождения искомой записи доступ к ней осуществляется прямым

способом по номеру записи, поэтому этот способ организации индекса и называется индексно-прямым.

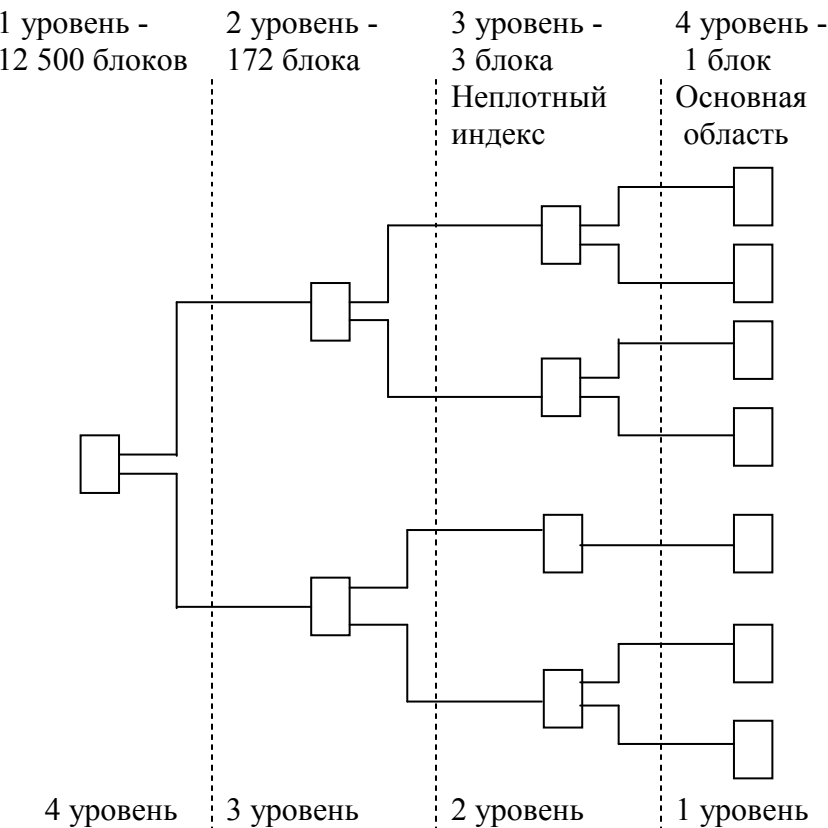


Рис. 5.5. Построенное В-дерево

В случае неплотного индекса после нахождения блока, в котором расположена искомая запись, поиск внутри блока требуемой записи происходит последовательным просмотром и сравнением всех записей блока. Поэтому способ индексации с неплотным индексом называется еще и индексно-последовательным.

5.4. Моделирование отношений «один-ко-многим» на файловых структурах

Отношение иерархии является типичным для баз данных, поэтому моделирование иерархических связей является типичным для физических моделей баз данных.

Для моделирования отношений 1:M (один-ко-многим) и M:M (многие-ко-многим) на файловых структурах используется принцип организации цепочек записей внутри файла и ссылки на номера записей для нескольких взаимосвязанных файлов.

Моделирование отношения 1:M с использованием одноподчиненных указателей. В этом случае связываются два файла, например F1 и F2, причем предполагается, что одна запись в файле F1 может быть связана с несколькими записями в файле F2. При этом файл F1 условно называется «основным», а файл F2 – «зависимым» или «подчиненным». Структура основного файла может быть условно представлена в виде трех областей.

«Основной файл» F1

Ключ	Запись	Ссылка-указатель на первую запись в «Подчиненном» файле, с которой начинается цепочка записей файла F2, связанных с данной записью файла F1
------	--------	---

В подчиненном файле также к каждой записи добавляется специальный указатель, в нем хранится номер записи, которая является следующей в цепочке записей «подчиненного» файла, связанной с одной записью «основного» файла. Таким образом, каждая запись «подчиненного» файла делится на две области: область указателя и область, содержащую собственно запись.

Структура записи «подчиненного» файла

Указатель на следующую запись в цепочке	Содержимое записи
---	-------------------

В качестве примера рассмотрим связь между преподавателями и занятиями, которые они ведут. В файле F1 приведен список преподавателей, а в файле F2 – список занятий, которые они ведут.

F1		
Номер записи	Ключ и остальная запись	Указатель
1	Ганцева Е.А.	1
2	Холопкина Л.В.	3
3	Тюрин С.В.	2

F2		
Номер записи	Указатель на следующую запись в цепочке	Содержимое записи
1	4	ВМ-011 Технология программирования
2	-	ВМ-001 Теория автоматов
3	5	ВМ-021 Программирование
4	-	ВМ-003 Программирование на Си
5	-	ВМ-011 Математическая логика

В этом случае содержимое двух взаимосвязанных файлов F1 и F2 может быть расшифровано следующим образом: первая запись в файле F1 связана с цепочкой записей файла F2, которая начинается с записи номер 1, следующая запись номер 4 (данная запись в цепочке – последняя, потому что запись 4 не имеет ссылки на следующую запись в цепочке). Аналогично можно расшифровать и остальные связи. Если провести интерпретацию связей на уровне предметной области, то можно утверждать, что преподаватель Е.А. Ганцева ведет предметы «Технология программирования» в группе ВМ-011, «Программива-

ние на Си» в группе ВМ-011. Аналогично могут быть расшифрованы и остальные взаимосвязанные записи.

5.5. Инвертированные списки

До сих пор рассматривались структуры данных, которые использовались для ускорения доступа по первичному ключу. Однако достаточно часто в базах данных требуется проводить операции доступа по вторичным ключам. Вторичным ключом является набор атрибутов, которому соответствует набор искоемых записей. Это означает, что существует множество записей, имеющих одинаковые значения вторичного ключа.

Для обеспечения доступа по вторичным ключам используются структуры, называемые инвертированными списками, которые послужили основой организации – индексных файлов для доступа по вторичным ключам.

Инвертированный список в общем случае – это двухуровневая индексная структура. Здесь на первом уровне находится файл или часть файла, в которой упорядоченно расположены значения вторичных ключей. Каждая запись с вторичным ключом имеет ссылку на номер первого блока в цепочке блоков, содержащих номера записей с заданным значением вторичного ключа. На втором уровне находится цепочка блоков, содержащих номера записей, имеющих одно и то же значение вторичного ключа. При этом блоки второго уровня упорядочены по значениям вторичного ключа. И наконец, на третьем уровне находится собственно основной файл.

Механизм доступа к записям по вторичному ключу при подобной организации состоит в следующем:

- на первом шаге в области первого уровня ищут заданное значение вторичного ключа;
- на втором шаге по ссылке считывают блоки второго уровня, содержащие номера записей с заданным значением вторичного ключа;

- на третьем шаге прямым доступом считывают из основного файла содержимое всех записей с заданным значением вторичного ключа.

Для одного основного файла может быть создано несколько инвертированных списков по разным вторичным ключам. Организация вторичных списков позволяет ускорить поиск записей с заданным значением вторичного ключа.

Рассмотрим модификацию основного файла. При модификации основного файла происходит следующее:

- изменяется запись основного файла;
- исключается старая ссылка на предыдущее значение вторичного ключа;
- добавляется новая ссылка на новое значение вторичного ключа.

Два последних шага выполняются для всех вторичных ключей, по которым созданы инвертированные списки. Такой процесс требует гораздо больше временных затрат, чем просто изменение содержимого записи основного файла без поддержки всех инвертированных списков.

Поэтому не следует безусловно утверждать, что введение индексных файлов (в том числе и инвертированных списков) всегда ускоряет обработку информации в базе данных. Если база данных постоянно изменяется, дополняется, модифицируется содержимое записей, то наличие большого количества инвертированных списков или индексных файлов по вторичным ключам может резко замедлить процесс обработки информации.

Можно придерживаться следующей позиции: если база данных достаточно стабильна и ее содержимое практически не меняется, то построение вторичных индексов действительно может ускорить процесс обработки информации.

5.6. Модели бесфайловой организации данных

Файловая структура и система управления файлами являются прерогативой операционной среды, поэтому принципы

обмена данными подчиняются законам операционной системы. По отношению к базам данных эти принципы могут быть далеки от оптимальности. СУБД подчиняется несколько иным принципам и стратегиям управления внешней памятью, чем те, которые поддерживают операционные среды для большинства пользовательских процессов или задач.

Это и послужило причиной того, что СУБД взяли на себя непосредственное управление внешней памятью. При этом пространство внешней памяти предоставляется СУБД полностью для управления, а операционная среда не получает непосредственного доступа к этому пространству.

Физическая организация современных баз данных является наиболее закрытой, она определяется как коммерческая тайна для большинства поставщиков коммерческих СУБД. И здесь не существует никаких стандартов, поэтому в общем случае каждый поставщик создает свою уникальную структуру и пытается обосновать ее наилучшие качества по сравнению со своими конкурентами. Физическая организация является в настоящий момент наиболее динамичной частью СУБД. Расширение возможностей устройств внешней памяти, увеличение объемов оперативной памяти изменяют сами принципы организации физических структур данных. И можно предположить, что и в дальнейшем эта часть современных СУБД будет постоянно меняться. Рассмотрим наиболее общие принципы и тенденции, существующие в организации физического хранения и обработки данных при бесфайловой организации данных.

При распределении дискового пространства рассматриваются две схемы структуризации: физическая, которая определяет хранимые данные, и логическая, которая определяет некоторые логические структуры, связанные с концептуальной моделью данных (рис. 5.6).

Определим некоторые понятия, используемые в указанной классификации.

Чанк (chunk) – представляет собой часть диска, физическое пространство на диске, которое ассоциировано одному процессу обработки данных).

Чанком может быть назначено неструктурированное устройство, часть этого устройства, блочно-ориентированное устройство или просто файл UNIX.

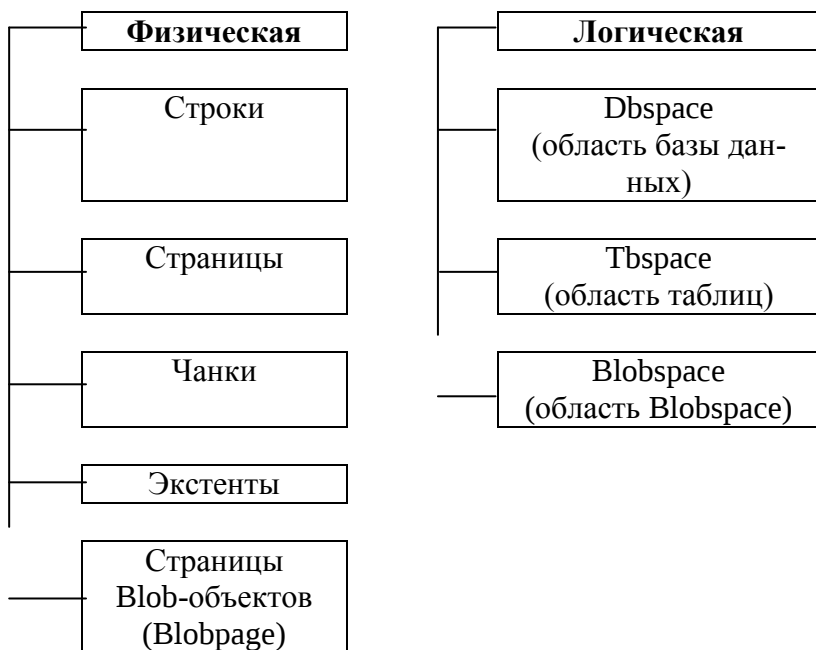


Рис. 5.6. Классификация объектов при статичной организации физической модели данных

Чанк характеризуется маршрутным именем, смещением (от физического начала устройства до начальной точки на устройстве, которая используется как чанк), размером, заданным в Кбайтах или Мбайтах.

При использовании блочных устройств и файлов величина смещения считается равной нулю.

Логические единицы образуются совокупностью экстен-тов, то есть таблица моделируется совокупностью экстен-тов.

Экстен-т – это непрерывная область дисковой памяти.

Для моделирования каждой таблицы используется два ти-па экстен-тов: первый и последующие. Первый экстен-т задается при создании нового объекта типа таблица, его размер задается при создании. EXTENTSIZE – размер первого экстен-та. NEXT SIZE – размер каждого следующего экстен-та.

Минимальный размер экстен-та в каждой системе свой, но в большинстве случаев он равен 4 страницам, максимальный – 2 Гбайтам. Новый экстен-т создается после заполнения предыду-щего и связывается с ним специальной ссылкой, которая распо-лагается на последней странице экстен-та. В ряде систем экстен-ты называются сегментами, но фактически эти понятия эквива-лентны.

При динамическом заполнении БД данными применяется специальный механизм адаптивного определения размера экс-тен-тов. Внутри экстен-та идет учет свободных страниц. Между экстен-тами, которые располагаются друг за другом без проме-жутков, производится своеобразная операция конкатенации, которая просто увеличивает размер первого экстен-та.

Механизм удвоения размера экстен-та: если число выде-ляемых экстен-тов для процесса растет в пропорции, кратной 16, то размер экстен-та удваивается каждые 16 экстен-тов. Напри-мер, если размер текущего экстен-та 16 Кбайт, то после запол-нения 16 экстен-тов данного размера размер следующего будет увеличен до 32 Кбайт.

Совокупность экстен-тов моделирует логическую единицу – таблицу-отношение (tblspace).

Экстен-ты состоят из четырех типов страниц: страницы данных, страницы индексов, битовые страницы и страницы blob-объектов. Blob – это сокращение Binary Large Object и соот-ветствует неструктурированным данным. В ранних СУБД такие данные относились к типу Мемо. В современных СУБД к этому типу относятся неструктурированные большие текстовые дан-

ные, картинки, наборы машинных кодов. Для СУБД важно знать, что этот объект надо хранить целиком, что размеры этих объектов от записи к записи могут резко отличаться и этот размер в общем случае неограничен.

Основной единицей осуществления операций обмена (ввода-вывода) является страница данных. Все данные хранятся постранично. При табличном хранении данные на одной странице являются однородными, то есть страница может хранить только данные или только индексы. Все страницы данных имеют одинаковую структуру, представленную на рис. 5.7.

Заголовок страницы (24 байта)
Содержание...
Слоты

Рис. 5.7. Обобщенная структура страницы данных

Слот – это 4-байтовое слово, 2 байта соответствуют смещению строки на странице и 2 байта – длина строки. Слоты характеризуют размещение строк данных на странице. На одной странице хранится не более 255 строк. В базе данных каждая строка имеет уникальный идентификатор в рамках всей базы данных, часто называемый RowID – номер строки, он имеет размер 4 байта и состоит из номера страницы и номера строки на странице. Под номер страницы отводится 3 байта, поэтому при такой идентификации возможна адресация к 16 777 215 страницам.

При упорядочении строк на страницах не происходит физического перемещения строк, все манипуляции происходят со слотами.

При переполнении страниц создается специальный вид страниц, называемых страницами остатка. Строки, не уместившиеся на основной странице, связываются с ней с помощью ссылок-указателей, которые содержат номер страницы и номер слота на странице.

Страницы индексов организованы в виде В-деревьев.

Страницы blob предназначены для хранения слабоструктурированной информации, содержащей тексты большого объема, графическую информацию, двоичные коды. Эти данные рассматриваются как потоки байтов произвольного размера, в страницах данных делаются ссылки на эти страницы.

Битовые страницы служат для трассировки других типов страниц. В зависимости от трассируемых страниц битовые страницы строятся по 2-битовой или 4-битовой схеме. 4-битовые страницы служат для хранения сведений о столбцах типа Varchar, Byte, Text, для остальных типов данных используются 2-битовые страницы.

Битовая структура трассирует 32 страницы. Каждая битовая структура представлена двумя 4-байтными словами. Каждая *i*-я позиция описывает одну *i*-ю страницу. Сочетание разрядов в *i*-х позициях двух слов обозначает состояние данной страницы: ее тип и занятость.

При обработке данных СУБД организует специальные структуры в оперативной памяти, называемые разделяемой памятью, и специальные структуры во внешней памяти, называемые журналами транзакций. Разделяемая память служит для кэширования данных при работе с внешней памятью с целью сокращения времени доступа, кроме того, разделяемая память служит для эффективной поддержки режимов одновременной параллельной работы пользователей с базой данных.

Журнал транзакций служит для управления корректным выполнением транзакций. В современных СУБД журнал транзакций обычно хранится в отдельном файле.

6. ЗАЩИТА ИНФОРМАЦИИ В БАЗАХ ДАННЫХ

6.1. Общие подходы к обеспечению безопасности данных

В современных СУБД поддерживается один из двух наиболее общих подходов к вопросу обеспечения безопасности

данных: избирательный подход и обязательный подход. В обоих подходах система безопасности может быть создана как для всей базы данных, так и для любого ее объекта.

Эти два подхода отличаются следующими свойствами

В случае избирательного управления некоторый пользователь обладает различными правами (привилегиями или полномочиями) при работе с данными объектами. Разные пользователи могут обладать разными правами доступа к одному и тому же объекту. Избирательные права характеризуются значительной гибкостью.

В случае обязательного управления каждому объекту данных присваивается некоторый классификационный уровень, а каждый пользователь обладает некоторым уровнем доступа. При таком подходе доступом к определенному объекту данных обладают только пользователи с соответствующим уровнем доступа.

Для реализации избирательного принципа предусмотрены следующие методы. В базу данных вводится новый тип объектов БД – это пользователи. Каждому пользователю в БД присваивается уникальный идентификатор. Для дополнительной защиты каждый пользователь снабжается также уникальным паролем, причем если идентификаторы пользователей в системе доступны системному администратору, то пароли пользователей хранятся чаще всего в специальном кодированном виде и известны только самим пользователям.

Пользователи могут быть объединены в специальные группы пользователей. Один пользователь может входить в несколько групп. В стандарте вводится понятие группы PUBLIC, для которой должен быть определен минимальный стандартный набор прав. По умолчанию предполагается, что каждый вновь создаваемый пользователь, если специально не указано иное, относится к группе PUBLIC.

Привилегии или полномочия пользователей или групп – это набор действий (операций), которые они могут выполнять над объектами БД.

В последних версиях ряда коммерческих СУБД появилось понятие «роли». Роль – это поименованный набор полномочий. Существует ряд стандартных ролей, которые определены в момент установки сервера баз данных. И имеется возможность создавать новые роли, группируя в них произвольные полномочия. Введение ролей позволяет упростить управление привилегиями пользователей. Введение ролей не связано с конкретными пользователями, поэтому роли могут быть определены и сконфигурированы до того, как определены пользователи системы.

Пользователю может быть назначена одна или несколько ролей.

Объектами БД, подлежащими защите, являются все объекты, хранимые в БД: таблицы, представления, хранимые процедуры, триггеры. Для каждого типа объектов есть свои действия, поэтому для каждого типа объектов могут быть определены разные права доступа.

Для обеспечения безопасности баз данных необходимо поддерживать два фундаментальных принципа: *проверку полномочий* и *проверку подлинности* (аутентификацию).

Проверка полномочий основана на том, что каждому пользователю или процессу информационной системы соответствует набор действий, которые он может выполнять по отношению к определенным объектам.

Проверка подлинности означает достоверное подтверждение того, что пользователь или процесс, пытающийся выполнить санкционированное действие, действительно тот, за кого он себя выдает.

6.2. Назначение и проверка полномочий, проверка подлинности

Система назначения полномочий имеет иерархический характер. Самыми высокими правами и полномочиями обладает системный администратор или администратор сервера БД. Тра-

диционно только этот тип пользователя может создавать других пользователей и наделять их определенными полномочиями.

СУБД в своих системных каталогах хранит как описание самих пользователей, так и описание их привилегий по отношению ко всем объектам.

Далее схема предоставления полномочий строится по следующему принципу. Каждый объект в БД имеет владельца – пользователя, который создал данный объект. Владелец объекта обладает всеми правами-полномочиями на данный объект, в том числе он имеет право предоставлять другим пользователям полномочия по работе с данным объектом или забирать у пользователей ранее предоставленные полномочия.

В ряде СУБД вводится следующий уровень иерархии пользователей – это администратор БД. В этих СУБД один сервер может управлять множеством СУБД (например, MS SQL Server, Sybase).

В СУБД Oracle применяется однобазовая архитектура, поэтому там вводится понятие подсхемы – части общей схемы БД и вводится пользователь, имеющий доступ к подсхеме.

Проверка полномочий осуществляется ядром СУБД при выполнении каждой операции. Логически для каждого пользователя и каждого объекта в БД как бы строится некоторая условная матрица, где по одному измерению расположены объекты, а по другому – пользователи. На пересечении каждого столбца и каждой строки расположены перечень разрешенных операций для данного пользователя над данным объектом. С первого взгляда кажется, что эта модель проверки достаточно устойчивая. Но сложность возникает тогда, когда используется косвенное обращение к объектам. Например, пользователю User_N не разрешен доступ к таблице Tab1, но этому пользователю разрешен запуск хранимой процедуры SP_N, которая делает выборку из этого объекта. По умолчанию все хранимые процедуры запускаются под именем их владельца.

Такие проблемы должны решаться организационными методами. При разрешении доступа некоторых пользователей не-

обходимо помнить о возможности косвенного доступа. В любом случае проблема защиты информации никогда не была чисто технической задачей, это комплекс организационно-технических мероприятий, которые должны обеспечить максимальную конфиденциальность информации, хранимой в БД.

При работе в сети существует проблема ***проверки подлинности полномочий***. Эта проблема состоит в следующем. Допустим, процессу 1 даны полномочия по работе с БД, а процессу 2 такие полномочия не даны. Тогда напрямую процесс 2 не может обратиться к БД, но он может обратиться к процессу 1 и через него получить доступ к информации из БД. Поэтому в безопасной среде должна присутствовать модель проверки подлинности, которая обеспечивает подтверждение заявленных пользователями или процессами идентификаторов.

Проверка полномочий приобрела еще большее значение в условиях массового распространения распределенных данных и вычислений. При существующем высоком уровне связности вычислительных систем необходимо контролировать все обращения к системе.

Проблемы проверки подлинности обычно относят к сфере безопасности коммуникаций и сетей. В целостной системе компьютерной безопасности, где четкое выполнение программы защиты информации обеспечивается за счет взаимодействия соответствующих средств в операционных системах, сетях, базах данных, проверка подлинности имеет прямое отношение к безопасности баз данных.

Модель безопасности, основанная на базовых механизмах проверки полномочий и проверки подлинности, не решает таких проблем, как украденные пользовательские идентификаторы и пароли или злонамеренные действия некоторых пользователей, обладающих полномочиями. Следовательно, программа обеспечения информационной безопасности должна охватывать не только технические области (такие как защита сетей, баз данных и операционных систем), но и проблемы физической защиты, надежности персонала (скрытые проверки), аудит, раз-

личные процедуры поддержки безопасности, выполняемые вручную или частично автоматизированные.

6.3. Средства защиты базы данных

Средства защиты БД в различных СУБД несколько отличаются друг от друга. На основе анализа современных СУБД Borland и Microsoft можно утверждать, что средства защиты БД условно делятся на две группы: основные и дополнительные.

К основным средствам защиты информации можно отнести следующие средства:

- парольной защиты;
- шифрования данных и программ;
- установления прав доступа к объектам БД;
- защиты полей и записей таблиц БД.

Парольная защита представляет простой и эффективный способ защиты БД от несанкционированного доступа. Пароли устанавливаются конечными пользователями или администраторами БД. Учет и хранение паролей производится самой СУБД. Обычно пароли хранятся в определенных системных файлах СУБД в зашифрованном виде. Поэтому просто найти и определить пароль невозможно. После ввода пароля пользователю СУБД предоставляются все возможности по работе с защищенной БД.

Шифрование данных (всей базы или отдельных таблиц) применяется для того, чтобы другие программы не могли прочитать данные. *Шифрование исходных текстов* программ позволяет скрыть от несанкционированного пользователя описание соответствующих алгоритмов.

В целях контроля использования основных ресурсов СУБД во многих системах имеются средства установления *прав доступа* к объектам БД. Права доступа определяют возможные действия над объектами. Владелец объекта (пользователь, создавший объект), а также администратор БД имеют все права.

Остальные пользователи к разным объектам могут иметь различные уровни доступа.

По отношению к таблицам в общем случае могут предусматриваться следующие права доступа:

- просмотр (чтение) данных;
- изменение (редактирование) данных;
- добавление новых записей;
- добавление и удаление данных;
- все операции, в том числе изменение структуры таблицы.

К данным, имеющимся в таблице, могут применяться меры защиты по отношению к отдельным полям и отдельным записям. В реляционных СУБД отдельные записи специально не защищаются.

Применительно к защите данных в полях таблиц можно выделить следующие уровни прав доступа:

- полный запрет доступа;
- только чтение;
- разрешение всех операций (просмотр, ввод новых значений, удаление и изменение).

По отношению к формам могут предусматриваться две основные операции: вызов для работы и разработка (вызов Конструктора). Запрет вызова Конструктора целесообразно делать для экранных форм готовых приложений, чтобы конечный пользователь случайно не испортил приложение. В самих экранных формах отдельные элементы могут быть тоже защищены. Например, некоторые поля исходной таблицы вообще могут отсутствовать или скрыты от пользователя, а некоторые поля – доступны только для просмотра.

Отчеты во многом похожи на экранные формы, за исключением следующего. Во-первых, они не позволяют изменять данные в таблицах, а во-вторых, основное их назначение – вывод информации на печать. На отчеты, так же, как и на экранные формы, может накладываться запрет на вызов средств их разработки.

Для исключения просмотра и модификации (случайной или преднамеренной) текстов программ, используемых в приложениях СУБД, помимо шифрации, может применяться их парольная защита.

К **дополнительным средствам защиты** БД можно отнести такие, которые нельзя прямо отнести к средствам защиты, но которые непосредственно влияют на безопасность данных. Это следующие средства:

- встроенные средства контроля значений данных в соответствии с типами;
- повышения достоверности вводимых данных;
- обеспечения целостности связей таблиц;
- организации совместного использования объектов БД в сети.

Редактируя БД, пользователь может случайно ввести такие значения, которые не соответствуют типу поля, в которое это значение вводится (например, ввод в числовое поле текстовой информации). В этом случае СУБД с помощью *средств контроля значений* блокирует ввод и сообщает пользователю об ошибке.

Средства повышения достоверности вводимых значений в СУБД служат для более глубокого контроля, связанного с семантикой обрабатываемых данных. Они обычно обеспечивают возможность при создании таблицы указывать следующие ограничения на значения: минимальное и максимальное значения, значение, принимаемое по умолчанию (если нет ввода), требование обязательного ввода; задание маски (шаблона) ввода и т.д.

Более совершенной формой организации контроля достоверности информации в БД является разработка хранимых процедур. Механизм хранимых процедур применяется в БД, размещенных на сервере. Сами хранимые процедуры представляют собой программы, алгоритмы которых предусматривают выполнение некоторых функций (в том числе контрольных) над данными. Процедуры хранятся вместе с данными и при необхо-

димости вызываются из приложений либо при наступлении некоторых событий в БД.

Решение прикладной задачи, как правило, требует выбора информации из нескольких таблиц. Таблицы в базе данных могут быть связаны. Функции поддержания *логической целостности связанных таблиц* берет на себя СУБД. Если СУБД не реализует эти функции, то ответственность за корректность связей возлагается на приложение.

Приведем пример возможных действий СУБД по контролю целостности связей таблиц. Пусть между двумя таблицами существует связь вида 1:М и, следовательно, одной записи основной таблицы может соответствовать несколько записей вспомогательной таблицы.

При вставке записей во вспомогательную таблицу система контролирует наличие соответствующих значений в поле связи основной таблицы. Если вводимое значение отсутствует в основной таблице, СУБД временно блокирует работу с новой записью и предлагает изменить значение или удалить запись целиком.

Удаление записей из дополнительных таблиц не контролируется. При удалении записи из основной таблицы происходит проверка. В случае, когда запись основной таблицы связана с несколькими записями дополнительной таблицы, возможны два варианта поведения. Не удалять основной записи, пока имеется хотя бы одна подчиненная запись (записи должен удалять пользователь), либо удалить основную запись и все подчиненные записи (каскадное удаление).

В распределенных информационных системах, работающих с базами данных, возникает проблема разрешения конфликтов между различными действиями над одними и теми же объектами (*совместного использования объектов* БД). Например, что делать в случае, когда один из пользователей локальной сети редактирует БД, а другой хочет изменить ее структуру? Для таких ситуаций в СУБД должны быть предусмотрены механизмы разрешения конфликтов.

Обычно при одновременной работе нескольких пользователей в сети используются блокировки. *Блокировки* могут действовать на различные объекты БД и на отдельные элементы объектов. Блокировки объектов возникают, когда параллельно с использованием объекта предпринимается попытка входа в режим разработки этого же объекта. Применительно к таблицам баз данных дополнительные блокировки могут возникать при работе с отдельными записями или полями.

Блокировки бывают явные и неявные. Явные блокировки накладываются пользователем или приложением с помощью команд. Неявные блокировки организует сама система, чтобы избежать возможных конфликтов. Например, в случае попытки изменения структуры БД во время редактирования информации устанавливается запрет реструктурирования БД до завершения редактирования данных.

7. РАСПРЕДЕЛЕННЫЕ БАЗЫ ДАННЫХ

7.1. Организация базы данных в локальной сети

Работа на изолированном компьютере с небольшой базой данных в настоящий момент становится нехарактерной для большинства приложений. БД отражает информационную модель реальной предметной области, хранит большие объемы информации, которая постоянно увеличивается. Соответственно увеличивается количество приложений, работающих с единой базой данных. Компьютеры объединяются в локальные сети и осуществляют доступ к корпоративной базе данных общего пользования, расположенной на сервере.

В общем случае режимы использования БД можно представить в следующем виде (рис. 7.1).

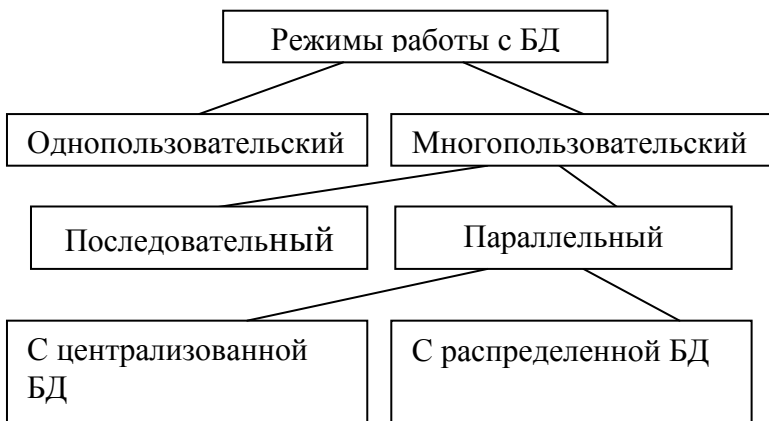


Рис. 7.1. Режимы работы с базой данных

Существует два варианта организации базы данных в локальной сети.

Первый вариант – **системы распределенной обработки данных**. Централизованная БД расположена на одной машине (сервере). К ней осуществляется параллельный доступ нескольких пользователей.

Второй вариант – **системы распределенных баз данных**. БД распределена на нескольких компьютерах, объединенных в сеть. К БД возможен параллельный доступ нескольких пользователей.

Распределенная база данных (РБД) – система логически интегрированных и территориально распределенных БД, языковых, программных, технических и организационных средств, предназначенных для создания, ведения и обработки информации.

К достоинствам распределенных баз данных (РБД) можно отнести следующее:

- соответствие структуры РБД структуре организации;
- гибкое взаимодействие локальных БД;
- непосредственный доступ к информации, снижение стоимости передачи данных;

- малое время отклика за счет распараллеливания процессов, высокая надежность;
- возможность распределения информации в соответствии с активностью ее использования;
- независимая параллельная разработка локальных БД.

Вместе с тем РБД обладают более сложной структурой, что вызывает появление дополнительных проблем:

- избыточность данных;
- несогласованность данных во времени;
- необходимость согласования процессов обновления и запросов;
- использование телекоммуникационных ресурсов;
- стандартизация общего интерфейса;
- реализация взаимодействия локальных БД;
- усложнение защиты данных.

Локальные базы данных, объединяемые в распределенную базу данных, могут быть:

- гомогенными (однородными), чаще всего реляционными, созданными с использованием СУБД одного производителя;
- гетерогенными (неоднородными), построенными с использованием разных моделей данных и созданных с использованием СУБД разных производителей.

Крупнейший специалист в области разработки теории баз данных Дейт К. сформулировал 12 правил для распределенных баз данных:

- локальная автономность;
- отсутствие опоры на центральный узел;
- непрерывное функционирование (развитие) РБД;
- независимость РБД от расположения локальных БД;
- независимость от фрагментации данных;
- независимость от репликации (дублирования) данных;
- обработка распределенных запросов;
- обработка распределенных транзакций;
- независимость от типа оборудования;

- независимость от операционной системы;
- независимость от сетевой архитектуры;
- независимость от типа СУБД.

7.2. Модели архитектуры клиент-сервер

При построении распределенных ИС, работающих с БД, широко используется архитектура клиент-сервер. Ее основу составляют принципы организации взаимодействия клиента и сервера при управлении БД.

Архитектура клиент-сервер – структура локальной сети, в которой применено распределенное управление сервером и рабочими станциями (клиентами) для максимально эффективного использования вычислительных мощностей.

Сервер – узловая станция компьютерной сети, предназначенная в основном для хранения данных коллективного пользования и для обработки в ней запросов, поступающих от пользователей других узлов.

Клиент – компьютер, обращающийся к совместно используемым ресурсам, которые представляются другим компьютером (сервером).

Структура распределенной ИС, построенной по архитектуре клиент-сервер с использованием сервера баз данных, показана на рис. 7.2.

При такой архитектуре сервер баз данных обеспечивает выполнение основного объема обработки данных. Формируемые пользователем или приложением запросы поступают к серверу БД в виде инструкций языка SQL. Сервер баз данных выполняет поиск и извлечение нужных данных, которые затем передаются на компьютер пользователя. Преимуществом такого подхода является заметно меньший объем передаваемых данных.

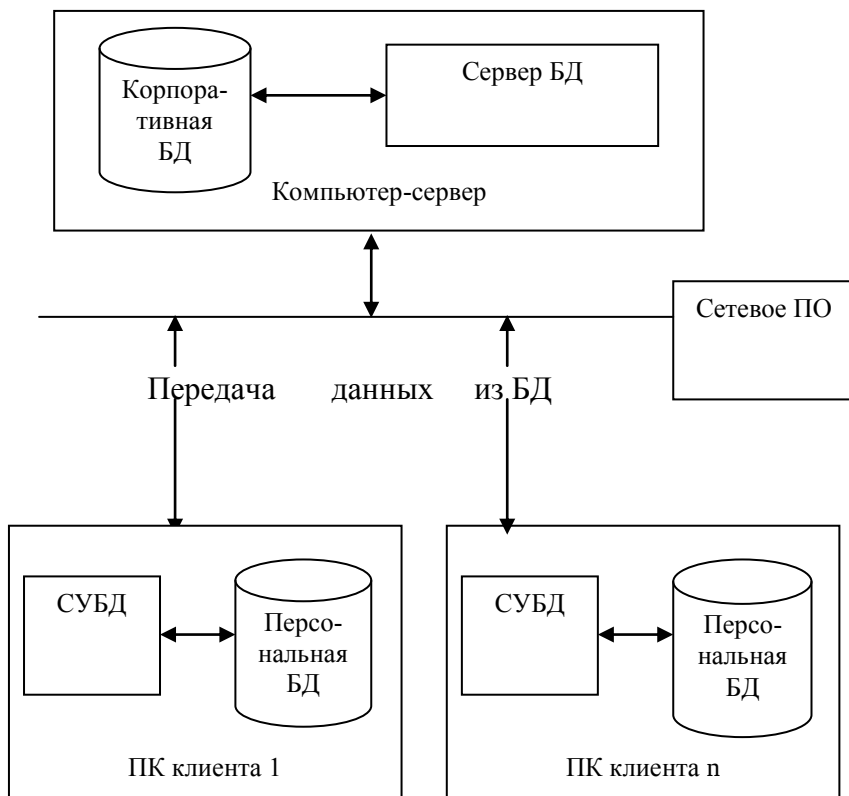


Рис. 7.2. Реализация клиент-серверной архитектуры

Для создания и управления персональными БД и приложениями, работающими с ними, используются СУБД, такие как Access и Visual FoxPro фирмы Microsoft, Paradox фирмы Borland.

Корпоративная БД создается, поддерживается и функционирует под управлением сервера БД, например, Microsoft SQL Server, Oracle Server, InterBase (FireBird), Sybase, Informix.

В зависимости от размеров организации и особенностей решаемых задач информационная система может иметь одну из следующих конфигураций:

- компьютер-сервер, содержащий корпоративную и персональные базы;
- компьютер-сервер и персональные компьютеры с ПБД;
- несколько компьютеров-серверов и персональных компьютеров с ПБД.

Использование архитектуры клиент-сервер дает возможность постепенного наращивания информационной системы предприятия, во-первых, по мере развития предприятия; во-вторых, по мере развития самой информационной системы.

Разделение общей БД на корпоративную БД и персональные БД позволяет уменьшить сложность проектирования БД по сравнению с централизованным вариантом, а значит снизить вероятность ошибок при проектировании и стоимость проектирования.

Согласно Эталонной модели Архитектуры открытых систем OSI функция управления БД относится к прикладному уровню.

СУБД как программа, поддерживающая интерфейс с пользователем, реализует следующие основные функции:

- управление данными, находящимися в базе;
- обработка информации с помощью прикладных программ;
- представление информации в удобном для пользователя виде.

При размещении СУБД в сети возможны различные варианты распределения функций по узлам сети. В зависимости от числа узлов сети, между которыми выполняется распределение функций СУБД, можно выделить двухзвенные и трехзвенные модели. Место разрыва функций соединяется коммуникационными элементами (средой передачи информации в сети).

Двухзвенные модели соответствуют распределению функций СУБД между двумя узлами сети. Компьютер (узел сети), на котором обязательно присутствует функция управления данными, называют компьютером-сервером. Компьютер, близ-

кий к пользователю и обязательно занимающийся вопросами представления информации, называют компьютером-клиентом.

Наиболее типичными вариантами разделения функций между компьютером-сервером и компьютером-клиентом являются следующие.

1. Распределенное представление:

- компьютер-сервер – управление данными, обработка, представление;

- компьютер-клиент – представление.

2. Удаленное представление:

- компьютер-сервер – управление данными, обработка;

- компьютер-клиент – представление.

3. Распределенная функция:

- компьютер-сервер – управление данными, обработка;

- компьютер-клиент – обработка, представление.

4. Удаленный доступ к данным

- компьютер-сервер – управление данными;

- компьютер-клиент – обработка, представление.

5. Распределенная БД

- компьютер-сервер – управление данными;

- компьютер-клиент – управление данными, обработка, представление.

Перечисленные способы распределения функций в системах с архитектурой клиент-сервер иллюстрируют различные варианты: от мощного сервера, когда практически вся работа производится на нем, до мощного клиента, когда большая часть функций выполняется на рабочей станции, а сервер обрабатывает поступившие к нему по сети SQL-вызовы.

Трехзвенная модель распределения функций представляет собой типовой вариант, при котором каждая из трех функций приложения (управление данными, обработка, представление) реализуется на отдельном компьютере.

Данная модель имеет название – **модель сервера приложений**, или **AS-модель** (Application Server) и показана на рис. 7.3.



Рис. 7.3. Трехзвенная модель сервера приложений

Согласно трехзвенной AS-модели процесс, отвечающий за организацию диалога с конечным пользователем, реализует функции представления информации и взаимодействует с компонентом приложения.

Компонент приложения, располагаясь на отдельном компьютере, в свою очередь, связан с компонентом управления данными.

Центральным звеном AS-модели является сервер приложений. На сервере приложений реализуется несколько прикладных функций, каждая из которых оформлена как служба предоставления услуг всем требующим этого программам. Серверов приложений может быть несколько, причем каждый из них предоставляет свой вид сервиса. Любая программа, запрашивающая услугу у сервера приложений, является для него клиентом. Поступающие от клиентов к серверам запросы помещаются в очередь, из которой выбираются в соответствии с некоторыми правилами, например, по приоритетам.

Компонент, реализующий функции представления и являющийся клиентом для сервера приложений, в этой модели трактуется более широко. Он может служить для организации интерфейса с конечным пользователем, обеспечивать прием

данных от устройств, например, датчиков, или быть произвольной программой.

Достоинством AS-модели является гибкость и универсальность вследствие разделения функций приложения на три независимые составляющие. Во многих случаях эта модель оказывается более эффективной по сравнению с двухзвенными. Основной недостаток модели – более высокие затраты ресурсов компьютеров на обмен информацией между компонентами приложения по сравнению с двухзвенными моделями.

Модель монитора транзакций. AS-модель описывает взаимодействие трех основных элементов: клиента, сервера приложений и сервера БД, но не затрагивает вопросы организации функционирования программного обеспечения при обработке информации, в частности при выполнении транзакций. Для преодоления этого недостатка предложена модель монитора транзакций.

Мониторы обработки транзакций (Transaction Processing Monitor – TPM), или мониторы транзакций, представляют собой программные системы категории промежуточного слоя (Middleware), обеспечивающие эффективное управление информационно-вычислительными ресурсами в распределенной вычислительной системе. Основное их назначение – организация гибкой, открытой среды для разработки и управления мобильными приложениями, оперативно обрабатывающими распределенные транзакции. Применение монитора транзакций наиболее эффективно в гетерогенных вычислительных системах.

Принципы организации обработки информации с помощью монитора транзакций описываются моделью монитора транзакций X / Open DTP (Distributed Transaction Processing – обработка распределенных транзакций). Эта модель включает в себя три объекта: прикладную программу, менеджер ресурсов (Resource Manager – RM) и монитор, или менеджер транзакций (Transaction Manager – TM).

В качестве прикладной программы может выступать произвольная программа-клиент.

Менеджер ресурсов RM выполняет функции сервера ресурса некоторого вида. Прикладная программа взаимодействует с RM с помощью набора специальных функций либо посредством операторов SQL (когда сервером является сервер БД). Функции менеджера ресурсов обычно выполняют серверы БД.

В задачах организации управления обработкой распределенных транзакций (транзакций, затрагивающих программные объекты вычислительной сети) ТМ взаимодействует с RM, который должен поддерживать протокол двухфазной фиксации транзакций и удовлетворять стандарту X / Open XA. Примерами СУБД, поддерживающих протокол двухфазной фиксации транзакций, являются Oracle 7.0, Open INGRES, Informix-Online 5.0.

Понятие транзакции в ТРМ (монитор транзакций) несколько шире, чем в СУБД, где транзакция включает в себя элементарные действия над данными базы. Здесь транзакция может охватывать и другие необходимые действия: передачу сообщения, запись информации в файл, опрос датчиков и т.д. Транзакции, которые поддерживают ТРМ, называют также прикладными или бизнес-транзакциями.

Для **разработчиков приложений** мониторы обработки транзакций ТРМ предоставляют удобства, связанные с возможностью декомпозиции приложений по нескольким функциональным уровням со стандартными интерфейсами, что позволяет создавать легко модифицируемые ИС со стройной архитектурой.

Для **пользователей распределенных систем** ТРМ позволяют улучшить показатели пропускной способности и времени отклика, снизить стоимость обработки данных в оперативном режиме на основе эффективной организации вычислительного процесса.

Администраторы распределенных систем, имея ТРМ, получают возможность легкого масштабирования ИС и увеличения производительности обработки информации. Без оста-

новки серверов приложений в любое время может быть добавлен, например, компьютер или менеджер ресурсов.

7.3. Управление распределенными данными

С управлением данными в распределенных системах связаны следующие две группы проблем: поддержка соответствия БД вносимым изменениям и обеспечение совместного доступа нескольких пользователей к общим данным.

Поддержка соответствия БД вносимым изменениям. В современных распределенных системах информация может храниться централизованно или децентрализованно. В первом случае проблемы идентичности представления информации для всех пользователей не существует, так как все последние изменения хранятся в одном месте. На практике информация может изменяться одновременно в нескольких узлах распределенной вычислительной системы. В этом случае возникает проблема контроля за всеми изменениями информации и предоставления ее в достоверном виде всем пользователям.

Существует две основные технологии децентрализованного управления БД: распределенных БД (Distributed Database) и тиражирования, или репликации, БД (Data Replication).

Распределенная БД состоит из нескольких фрагментов, размещенных на разных узлах сети и, возможно, управляемых разными СУБД. С точки зрения программ и пользователей, обращающихся к распределенной БД, последняя воспринимается как единая локальная БД (рис. 7.4).

Информация о местоположении каждой из частей распределенной БД и другая служебная информация хранится в так называемом глобальном словаре данных. В общем случае этот словарь может храниться на одном из узлов или тоже быть распределенным.

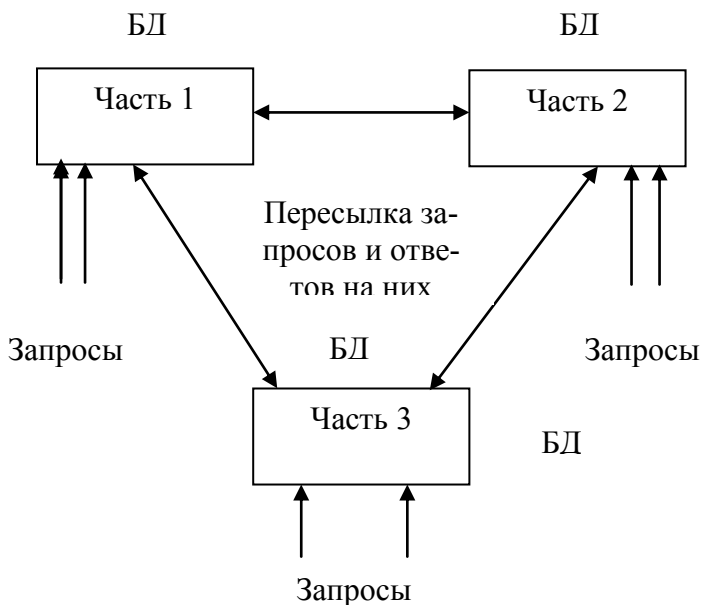


Рис. 7.4 Модель распределенной БД

Для обеспечения корректного доступа к распределенной БД в современных системах чаще всего применяется протокол (метод) двухфазной фиксации транзакций (two-phase commit). Сущность этого метода состоит в двухэтапной синхронизации выполняемых изменений на всех задействованных узлах. На первом этапе в узлах сети производятся изменения в их БД, о чем посылаются уведомления компоненту системы, управляющему обработкой распределенных транзакций.

На втором этапе, получив от всех узлов сообщения о правильности выполнения операций (что свидетельствует об отсутствии сбоев и отказов аппаратно-программного обеспечения), управляющий компонент выдает всем узлам команду фиксации изменений. После этого транзакция считается завершенной, ее результат необратимым.

Основным достоинством модели распределенной БД является то, что пользователи всех узлов (при исправных коммуникационных средствах) получают информацию с учетом всех последних изменений. Второе преимущество состоит в экономном использовании внешней памяти компьютеров, что позволяет организовать БД больших объемов.

К недостаткам модели распределенной БД относится следующее: жесткие требования к производительности и надежности каналов связи, а также большие затраты коммуникационных и вычислительных ресурсов из-за их связывания на все время выполнения транзакций. При интенсивных обращениях к распределенной БД, большом числе взаимодействующих узлов, низкоскоростных и ненадежных каналах связи обработка запросов по этой схеме становится практически невозможной.

Модель тиражирования данных, в отличие от технологии распределенных БД, предполагает дублирование данных (создание точных копий) в узлах сети (рис. 7.5).

Данные всегда обрабатываются как обычные локальные. Поддержку идентичности копий друг другу в асинхронном режиме обеспечивает компонент системы, называемый репликатором (replicator). При этом между узлами сети могут передаваться как отдельные изменения, так и группы изменений. В течении некоторого времени копии БД могут отличаться друг от друга.

К основным достоинствам модели тиражирования БД (в сравнении с предыдущей моделью) относятся: более высокая скорость доступа к данным, так как они всегда есть в узле; существенное уменьшение передаваемого по каналам связи потока информации, поскольку происходит передача не всех операций доступа к данным, а только изменений в БД; повышение надежности механизмов доступа к распределенным данным, поскольку нарушение связи не приводит к потере работоспособности системы.

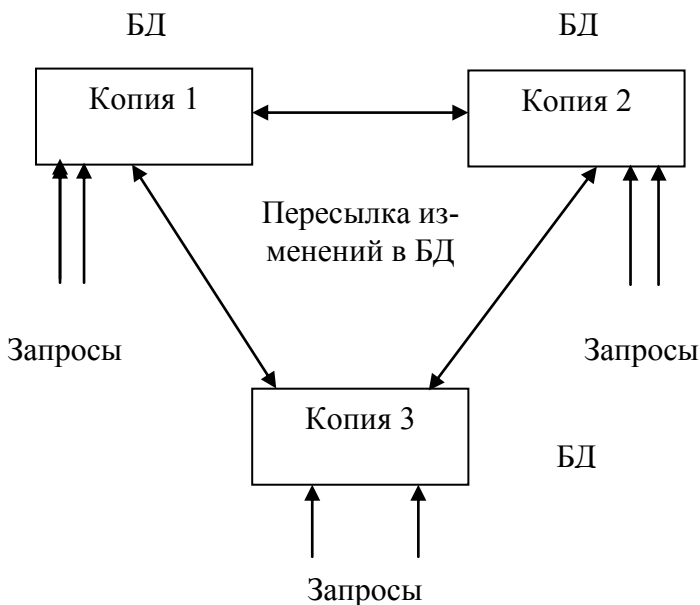


Рис. 7.5. Модель тиражирования БД

Основной недостаток модели тиражирования БД заключается в том, что на некотором интервале времени возможно «расхождение» копий БД. Если отмеченный недостаток не критичен для прикладных задач, то предпочтительно иметь схему с тиражированием БД.

Доступ к общим данным. При обслуживании обращений к общим данным средства управления БД должны обеспечивать по крайней мере два основных метода доступа: монопольный и коллективный.

Основными объектами доступа в различных системах могут быть целиком БД, отдельные таблицы, записи, поля записей, элементы приложения такие, как отчеты, запросы, экранные формы.

Монопольный доступ обычно используется в двух случаях:

во-первых, когда требуется исключить доступ к объектам со стороны других пользователей (например, при работе с конфиденциальной информацией);

во-вторых, когда производятся ответственные операции с БД, не допускающие других действий, например, изменение структуры БД.

В первом случае пользователь с помощью диалоговых средств СУБД или прикладной программы устанавливает явную блокировку. Во втором случае пользователь также может установить явную блокировку, либо положиться на СУБД. Последняя обычно автоматически устанавливает неявную (без ведома пользователя или приложения) блокировку, если это необходимо.

В режиме **коллективного доступа** полная блокировка на используемые объекты, как правило, не устанавливается. Коллективный доступ возможен, например, при одновременном просмотре таблиц. Попытки получить монопольный доступ к объектам коллективного доступа должны быть пресечены. Например, в ситуации когда один или несколько пользователей просматривают таблицу, а другой пользователь собирается удалить эту же таблицу.

Для организации коллективного доступа в СУБД применяется **механизм блокировок**. Суть блокировки состоит в том, что на время выполнения какой-либо операции в БД доступ к используемому объекту со стороны других потребителей временно запрещается или ограничивается. Например, при копировании таблицы она блокируется от изменения, хотя и разрешено просматривать ее содержимое.

Рассмотрим типичный набор блокировок. В конкретных программах схемы блокирования объектов могут отличаться от описываемых.

Выделим четыре вида блокировок, перечисленных в порядке убывания строгости ограничений на возможные действия:

полная блокировка;

блокировка от записи;
предохраняющая блокировка от записи;
предохраняющая полная блокировка.

Полная блокировка. Означает полное завершение всяких операций над основными объектами (таблицами, отчетами и экранными формами). Этот вид блокировок обычно применяется при изменении структуры таблицы.

Блокировка от записи. Накладывается в случаях, когда можно использовать таблицу, но без изменения ее структуры или содержимого. Такая блокировка применяется, например, при выполнении операции слияния данных из двух таблиц.

Предохраняющая блокировка от записи. Предохраняет объект от наложения на него со стороны других операций полной блокировки, либо блокировки от записи. Этот вид блокировки позволяет тому, кто раньше «захватил» объект, успешно завершить модификацию объекта. Предохраняющая блокировка от записи совместима с аналогичной блокировкой (предохраняющей блокировкой от записи), а также с предохраняющей полной блокировкой. Примером необходимости использования этой блокировки является режим совместного редактирования таблицы несколькими пользователями.

Предохраняющая полная блокировка. Предохраняет объект от наложения на него со стороны других операций только полной блокировки. Обеспечивает максимальный уровень совместного использования объектов. Такая блокировка может использоваться, например, для обеспечения одновременного просмотра несколькими пользователями одной таблицы. В группе пользователей, работающих с одной таблицей, эта блокировка не позволит никому изменить структуру общей таблицы.

При незавершенной операции с некоторым объектом и запросе на выполнение новой операции с этим же объектом производится попытка эти операции совместить. Совмещение возможно тогда, когда совместимыми оказываются блокировки, накладываемые конкурирующими операциями.

В отношении перечисленных выше четырех блокировок действуют следующие правила совмещения:

при наличии полной блокировки над объектом нельзя производить операции, приводящие хотя бы к одному из видов блокировок (полная блокировка несовместима ни с какой другой блокировкой);

блокировка от записи совместима с аналогичной блокировкой и предохраняющей полной блокировкой;

предохраняющая блокировка от записи совместима с обоими видами предохраняющих блокировок;

предохраняющая полная блокировка совместима со всеми блокировками, кроме полной.

Обычно в СУБД каждой из выполняемых с БД операций соответствует определенный вид блокировок, которую эта операция накладывает на объект. Пользователям современных СУБД, работающих в интерактивном режиме, не нужно помнить все тонкости механизма блокировки, поскольку система достаточно «разумно» осуществляет автоматическое блокирование во всех случаях, когда это требуется. При этом система сама стремится предоставить всем пользователям наиболее свободный доступ к объектам. При необходимости можно воспользоваться командными и языковыми средствами явного определения блокировок.

Тупики. Если не управлять доступом к совместно используемым объектам, то между потребителями ресурсов могут возникать тупиковые ситуации. Следует отличать понятие блокировки в смысле контроля доступа к объектам от блокировки в смысле тупикового события.

Существует два основных вида тупиков: взаимные (dead-lock) и односторонние (livelock).

Простейшим случаем **взаимного тупика** является ситуация, когда каждый из двух пользователей стремится захватить данные, уже захваченные другим пользователем (рис. 7.6). В этой ситуации пользователь-1 ждет освобождения ресурса N, в то время как пользователь-2 ожидает освобождения от захвата

ресурса М. Следовательно, никто из них не может продолжить работу.

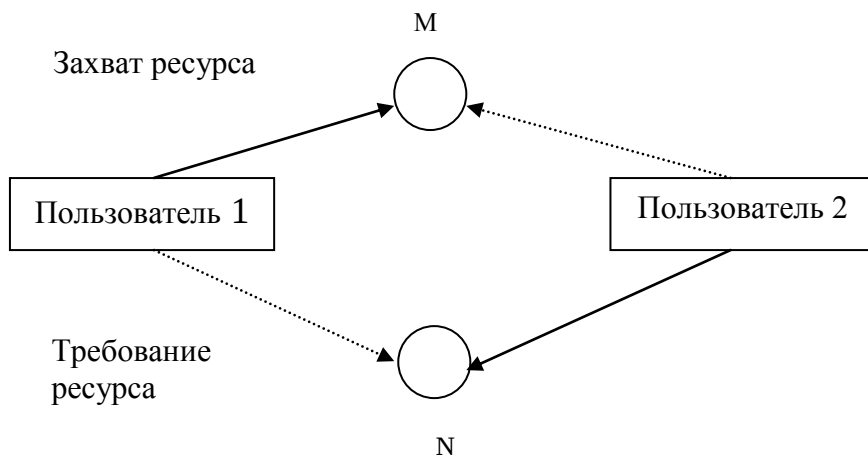


Рис. 7.6. Пример взаимного тупика

Возможны и более сложные ситуации, когда выполняются обращения трех и более пользователей к нескольким ресурсам.

Односторонний тупик возникает в случае требования получить монопольный доступ к некоторому ресурсу как только он станет доступным и невозможности удовлетворить это требование.

Системы управления распределенными БД, очевидно должны иметь соответствующие средства обнаружения или предотвращения конфликтов, а также разрешения возникающих конфликтов.

ЗАКЛЮЧЕНИЕ

Базы данных остаются динамично развивающимся теоретико-прикладным направлением.

Современные базы данных являются основой многочисленных информационных систем. Информация, накопленная в них, является чрезвычайно ценным материалом. В связи с этим интенсивно развиваются методы обработки баз данных, позволяющие извлечь из них дополнительные знания, обобщить имеющуюся информацию и обработать ее дополнительными способами.

Развитие теории баз данных идет по следующим направлениям.

Создание и использование **хранилищ данных** (Data Warehouse).

Для работы с «Хранилищами данных» наиболее значимым становится так называемый **интеллектуальный анализ данных (ИАД)**, или data mining – это процесс выявления значимых корреляций, образцов и тенденций в больших объемах данных.

В бизнес-приложениях наибольший интерес представляет интеграция методов интеллектуального анализа данных с **технологией оперативной аналитической обработки данных** (On-Line Analytical Processing, **OLAP**). OLAP использует многомерное представление агрегированных данных для быстрого доступа к важной информации и дальнейшего ее анализа.

Следующим направлением в развитии систем управления базами данных является **отказ от нормализации отношений**. Во многом нормализация отношений нарушает естественные иерархические связи между объектами, которые достаточно распространены в мире. Возможность сохранить иерархические связи на концептуальном (но не на физическом) уровне позволяет пользователям более естественно отражать семантику предметной области. В настоящий момент существуют теоретическое обоснование работы с ненормализованными отношениями и практические реализации подобных систем.

В сфере моделирования данных наиболее активно развиваются **объектно-ориентированные базы данных**.

Следующим направлением развития баз данных является появление так называемых **темпоральных баз данных**, то есть баз данных, чувствительных ко времени. Фактически БД моделирует состояние объектов предметной области в некоторый текущий момент времени. Однако в ряде прикладных областей необходимо исследовать именно изменение состояний объектов во времени. Если использовать чисто реляционную модель, то требуется строить и хранить дополнительно множество отношений, имеющих одинаковые схемы, отличающиеся временем существования или снятия данных. Гораздо перспективнее и удобнее для этого использовать специальные механизмы снятия срезов по времени для определенных объектов БД. Основным тезис темпоральных систем состоит в том, что для любого объекта данных, созданного в момент времени t_1 и уничтоженного в момент времени t_2 , в БД сохраняются (и доступны пользователям) все его состояния во временном интервале от t_1 до t_2 .

Еще одним из перспективных направлений развития баз данных является направление, связанное с объединением технологии экспертных систем и баз данных и развитием так называемых **дедуктивных баз данных**. Эти базы основаны на выявлении новых знаний из баз данных не путем реализации запросов или осуществления аналитической обработки, а путем использования правил вывода и построения цепочек применения этих правил для вывода ответов на запросы. Для этих баз данных существуют языки запросов, отличные от классического языка SQL. В экспертных системах знания экспертов хранятся в форме правил, чаще всего используются так называемые продукционные правила типа «если описание ситуации, то описание действия».

Одним из самых значительных направлений развития баз данных является **взаимодействие Web-технологий и баз данных**. В Интернете сосредоточены громадные объемы слабо-структурированной и разнообразной информации, которой тре-

буется эффективно управлять. Web представляет собой одну громадную базу данных. Однако до сих пор базы данных остаются на вторых ролях и не превратились в неотъемлемую часть инфраструктуры Web. В подавляющей части Web-узлов, особенно в тех, которые принадлежат провайдерам и держателям поисковых машин, технология баз данных не применяется. В небольших Web-узлах, как правило, используются статические HTML-страницы, хранящиеся в обычных файловых системах.

Однако наблюдаются тенденции, свидетельствующие о сближении Web-технологий и баз данных:

дизайнеры крупнейших Web-серверов с миллионами страниц содержимого постепенно перекладывают задачи управления страницами с файловых систем на системы баз данных;

системы баз данных используются в качестве серверов электронной коммерции;

ведущие Web-издатели исследуют возможности использования систем баз данных для хранения информационного наполнения, имеющего сложную природу.

В будущем статические HTML-страницы станут заменяться системами управления динамически формируемым содержанием.

Авторы Web-публикаций нуждаются в инструментах для быстрого и экономичного построения хранилищ данных, рассчитанных на сложные приложения. Это, в свою очередь, формирует требования к технологии баз данных, используемой для создания, управления, поиска и обеспечения безопасности содержимого Web-узлов.

Универсальность Web-клиента становится весьма привлекательной для разработчиков несложных приложений, которые смогут работать с базами данных. В этом случае не требуется установка каждого клиента, достаточно выслать код доступа и клиент автоматически может уже работать с базой данных. При этом клиент может работать как в локальной, так и в глобальной сети, если технология это позволяет. Это весьма удобно,

когда можно с любого рабочего места, имея соответствующий пароль, получить доступ к необходимым данным. Подобные системы называются системами, разработанными по Интранет-технологии, то есть технологии, использующей принципы технологий Интернета, но реализованные во внутренней локальной сети.

Для разработки Интернет-приложений, которые связаны с базами данных, широко используются новые средства программирования: это язык PERL, язык PHP (Personal Home Page Tools), язык Javascript и ряд других.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Советов Б.Я. Базы данных: теория и практика: учебник для бакалавров / Б.Я. Советов, В.В. Цехановский, В.Д. Чертовской. М.: Издательство Юрайт, 2012.
2. Дейт К.Дж. Введение в системы баз данных / К.Дж. Дейт. М.: Вильямс, 2000.
3. Конноли Т. БД: проектирование, реализация и сопровождение / Т. Конноли, К. Бегг, А. Строган. М.: Вильямс, 2000.
4. Харрингтон Дж. Проектирование объектно-ориентированных баз данных / Дж. Харрингтон. М.: Вильямс, 2000.
5. Хомоненко А.Д. Базы данных / А.Д. Хомоненко, В.М. Цыганков, М.Г. Мальцев. СПб.: КОРОНА принт, 2004.
6. Карпова Т.С. Базы данных: модели, разработка, реализация / Т.С. Карпова. СПб.: Питер, 2001.
7. Хансен Г. Базы данных: разработка и управление / Г. Хансен, Дж. Хансен. М.: БИНОМ, 2000.
8. Ульман Д. Введение в системы баз данных / Д. Ульман. М.: Лори, 2000.
9. Маклаков С.В. BPwin и ERwin. CASE-средства разработки информационных систем / С.В. Маклаков. М.: ДИАЛОГ-МИФИ, 2001.
10. Избачков Ю.С. Информационные системы / Ю.С. Избачков, В.Н. Петров. СПб.: Питер, 2005.

ОГЛАВЛЕНИЕ

Введение	3
1. Основные понятия и определения.....	4
1.1. Информационные системы и банк данных	4
1.2. Назначение и основные компоненты банка данных..	5
1.3. Архитектура базы данных. Физическая и логическая независимость данных.....	9
1.4. Системы управления базами данных	14
1.5. Оперативные и аналитические системы	20
1.6. Требования, предъявляемые к базам данных	27
2. Модели данных	30
2.1. Иерархическая модель данных	31
2.2. Сетевая модель	34
2.3. Реляционная модель	37
2.4. Постреляционная модель	38
2.5. Многомерная модель	41
2.6. Объектно-ориентированная модель	47
2.7. Объектно-реляционная модель данных	56
3. Реляционная модель данных.....	57
3.1. Основные определения.....	57
3.1.1. Определение отношения, домена, кортежа, реляционной базы данных, ключей.....	57
3.1.2. Классы отношений.....	61
3.1.3. Индексирование	63
3.1.4. Связи между отношениями (таблицами)	64
3.1.5. Обеспечение целостности данных	66
3.2. Операции реляционной алгебры	71
3.2.1. Основные понятия.....	71
3.2.2. Базовые теоретико-множественные операции реляционной алгебры	73
3.2.3. Специальные операции реляционной алгебры	77
3.3. Реляционное исчисление.....	81
3.4. Язык запросов по образцу QBE	85
3.5. Структурированный язык запросов SQL	87

3.5.1. История развития SQL.....	87
3.5.2. Общая характеристика языка.....	88
3.5.3. Структура SQL	90
3.5.4. Оператор выбора SELECT	93
3.5.5. Применение агрегатных функций и группировки.....	100
3.5.6. Раздел ORDER BY и ключевое слово TOP ...	107
3.5.7. Вложенные запросы.....	109
3.5.8. Внутренние и внешние объединения.....	111
3.5.9. Перекрестные запросы	116
3.5.10. Операторы манипулирования данными.....	117
3.5.11. Запросы на создание таблиц	120
3.5.12. Использование языка определения данных	121
4. Проектирование баз данных	129
4.1. Этапы проектирования БД.....	129
4.2. Проблемы проектирования реляционных баз данных.....	131
4.3. Нормализация отношений.....	135
4.4. Метод сущность-связь	146
4.5. Средства автоматизации проектирования	158
4.5.1. Основные определения.....	158
4.5.2. Модели жизненного цикла.....	159
4.5.3. Модели структурного проектирования.....	161
4.5.4. Объектно-ориентированные модели.....	167
4.5.5. Классификация CASE-средств	171
5. Физические модели баз данных.....	173
5.1. Файловые структуры, используемые в базах данных.....	173
5.2. Хешированные файлы	176
5.2.1. Стратегия разрешения коллизий с областью переполнения.....	177
5.2.2. Организация стратегии свободного замещения	179
5.3. Индексные файлы	180

5.3.1. Файлы с плотным индексом, или индексно-прямые файлы	181
5.3.2. Файлы с неплотным индексом, или индексно-последовательные файлы	186
5.3.3. Организация индексов в виде B-tree (B-деревьев)	188
5.4. Моделирование отношений «один-ко-многим» на файловых структурах.....	191
5.5. Инвертированные списки	193
5.6. Модели бесфайловой организации данных.....	194
6. Защита информации в базах данных.....	199
6.1. Общие подходы к обеспечению безопасности данных.....	199
6.2. Назначение и проверка полномочий, проверка подлинности	201
6.3. Средства защиты базы данных	204
7. Распределенные базы данных.....	208
7.1. Организация базы данных в локальной сети.....	208
7.2. Модели архитектуры клиент-сервер	211
7.3. Управление распределенными данными	218
Заключение	226
Библиографический список	230

Учебное издание

Сергеева Татьяна Ивановна
Сергеев Михаил Юрьевич

БАЗЫ ДАННЫХ: МОДЕЛИ ДАННЫХ,
ПРОЕКТИРОВАНИЕ, ЯЗЫК SQL

В авторской редакции

Подписано к изданию 17.09.2012.
Объем данных 1,2 Мб

ФГБОУ ВПО «Воронежский государственный
технический университет»
394026 Воронеж, Московский просп., 14