Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Магнитогорский государственный технический университет им. Г.И. Носова»

М.В. Махмутова

ВВЕДЕНИЕ В ТЕХНОЛОГИИ БАЗ ДАННЫХ

*Утверждено Редакционно-издательским советом университета
в качестве учебного пособия*

Магнитогорск
2015

УДК 681.142.1.01
ББК 3973.23

Рецензенты:

Кандидат технических наук,
заместитель директора по развитию,
ЗАО «Консом СКС»
Ю.Н. Волщуков

Кандидат технических наук,
доцент кафедры бизнес-информатики и информационных технологий,
ФГБОУ ВПО «Магнитогорский государственный технический
университет им. Г.И. Носова»
С.А. Повитухин

Махмутова М.В.
Введение в технологии баз данных [Электронный ресурс] : учебное пособие / Марина Владимировна Махмутова ; ФГБОУ ВПО «Магнитогорский государственный технический университет им. Г.И. Носова». – Электрон. текстовые дан. (1,81 Мб). – Магнитогорск : ФГБОУ ВПО «МГТУ», 2015. – 1 электрон. опт. диск (CD-R). – Систем. требования : IBM PC, любой, более l GHz ; 512 Мб RAM ; 100 Мб HDD ; MS Windows XP и выше ; Adobe Reader 8.0 и выше ; CD/DVD-ROM дисковод ; мышь. – Загл. с титул. экрана.

Учебное пособие «Введение в технологии баз данных» разработано в поддержку ряда дисциплин базовой части профессионального цикла, таких как «Базы данных», «Проектирование ИС», «Управление жизненным циклом ИС», «Разработка приложений» и др. для формирования компетенций выпускника по образовательным программам 230700.62 (09.03.03) «Прикладная информатика» и 080500.62 «Бизнес-информатика» как ИТ-специалиста в области разработки баз данных и информационных систем.

В учебном пособии рассмотрены основные понятия технологий баз данных, описаны функциональные возможности систем управления базами данных, представлены назначение и использование языка структурированных запросов – SQL, применение SQL-операторов. Уделено внимание проблемам стандартизации в области технологий баз данных. Подробно описаны различные методологические подходы к моделированию данных и соответствующие этим подходам инструментальные средства (AllFusion Data Modeler и Process Modeler компании CA). Рассмотрены этапы процесса проектирования базы данных, проблемы автоматизации проектирования базы данных. Изложены основы использования технологий WWW для доступа к базам данных. Предложенный теоретический материал сопровождается большим количеством примеров, предлагаются вопросы для самопроверки.

Учебное пособие ориентировано на студентов очной и заочной форм обучения, изучающих технологии баз данных, методологии моделирования данных, проблемы проектирования базы данных, проблемы стандартизации в области технологий баз данных.

УДК 681.142.1.01
ББК 3973.23

Содержание

ВВЕДЕНИЕ

В учебном пособии рассмотрены основные понятия технологий баз данных, описаны функциональные возможности систем управления базами данных, представлены назначение и использование языка структурированных запросов – SQL, применение SQL-операторов. Уделено внимание проблемам стандартизации в области технологий баз данных. Подробно описаны различные методологические подходы к моделированию данных и соответствующие этим подходам инструментальные средства (AllFusion Data Modeler и Process Modeler компании CA). Рассмотрены этапы процесса проектирования базы данных, проблемы автоматизации проектирования базы данных. Изложены основы использования технологий WWW для доступа к базам данных. Предложенный теоретический материал сопровождается большим количеством примеров, предлагаются вопросы для самопроверки.

Работа ориентирована на студентов очного и заочного отделения, изучающих технологии баз данных, методологии моделирования данных, проблемы проектирования базы данных, проблемы стандартизации в области технологий баз данных.

В первой главе учебного пособия описываются основные этапы истории развития технологий баз данных, стандарты в области баз данных, этапы жизненного цикла баз данных, основные понятия технологии баз данных. В качестве стандартов технологий баз данных рассматриваются стандарты языка реляционных систем баз данных спецификации версии языка SQL, стандарты объектных баз данных группы ODMG, стандарт архитектурной концепции СУБД ANSI/X3/SPARC. Знание стандартов позволит создать у будущих специалистов не только фундаментальные основы знаний технологий баз данных, но и сформировать критерии качества проектируемых баз данных.

Вторая глава посвящена рассмотрению основ систем управления базами данных: представлена архитектура СУБД в соответствии с уровнями описания данных, описаны функциональные возможности СУБД, дана характеристика основных типов систем управления базами данных, представлены характеристики современных серверных СУБД.

В качестве примера современных СУБД в третьей главе рассмотрены основные компоненты и вопросы управления доступом и обработкой данных реляционной системы управления базами данных, а именно, системы Microsoft SQL Server.

В четвертой главе рассмотрены назначение и особенности языка структурированных запросов – SQL, выполнение операторов, вопросы создания и использования объектов баз данных средствами SQL

Пятая глава учебного пособия посвящена методологическим основам моделирования данных. В этой главе автор формирует понятийный аппарат учебного курса, рассматривая основные понятия с точки зрения различных стандартов. Кроме этого, в этой главе рассмотрены основные методологические подходы к моделированию данных на разных уровнях абстракции в соответствии с трехуровневой архитектурой стандарта ANSI/X3/SPARC. В качестве примера структурного подхода к моделированию предметной области рассмотрена функциональная методика потоков данных и реализующий эту методику инструментарий Bpwin фирмы Computer Associates. В качестве примера методологии визуального моделирования данных рассмотрены концепция и семантика IDEF1X, и реализующий эту методику инструментарий AllFusion ERwin Data Modeler фирмы Computer Associates, рассмотрен объектный подход к моделированию данных с использованием UML.

Шестая глава учебного пособия посвящена проблемам собственно проектирования баз данных. Процесс проектирования базы данных авторы рассматривают, как этап детального проектирования информационной системы. Проект описывается с точки зрения трехсхемного подхода: концептуальное проектирование, логическое проектирование, физическое проектирование. Такой подход позволяет студентам глубже понять проблемы

создания БД, как основного компонента ИС. В этой главе также затронуто современное состояние средств автоматизации проектирования баз данных.

Последняя глава учебного пособия посвящена описанию основ использования технологий WWW для доступа к базам данных, представлен обзор технологий, общая схема работы, обозначены проблемы и перспективы.

В пособии строго соблюдается структура учебного текста, оформленного по разделам, главам, параграфам с равномерными объемами структурных единиц, которые соответствуют познавательным возможностям студентов. Предлагается система практических заданий и вопросов для самостоятельного изучения, которая формирует и развивает потребности студентов в дальнейшем самостоятельном изучении различных аспектов технологий баз данных.

ГЛАВА 1. ОСНОВЫ ТЕХНОЛОГИИ БАЗ ДАННЫХ

"История исследований систем баз данных — это, по сути, история развития приложений, достигших исключительной производительности и оказавших потрясающее влияние на экономику. В 70-е годы 20 века эта сфера была всего лишь областью фундаментальных научных исследований, то теперь на исследованиях баз данных основана целая индустрия информационных услуг, ежегодный бюджет которой только в США составляет 10 миллиардов долларов. Достижения в исследованиях баз данных стали основой фундаментальных разработок коммуникационных систем транспорта и логистики, финансового менеджмента, систем с базами знаний, методов доступа к научной литературе, а также большого количества гражданских и военных приложений. Они также послужили фундаментом значительного прогресса в ведущих областях науки — от информатики до биологии", — Зильбершац (Silbftischatz ct al., 1991).

Эта цитата взята из материалов семинара по системам баз данных и, по сути, является достаточным основанием для того, чтобы понять важность изучения систем с базами данных.

Появление баз данных стало самым важным достижением в области программного обеспечения. Базы данных лежат в основе информационных систем, и это коренным образом изменило характер работы многих организаций. С момента своего появления технология баз данных стала увлекательной областью деятельности, а также катализатором многих значительных достижений в области программного обеспечения.

Базы данных являются неотъемлемой частью нашей повседневной жизни. Можно рассматривать базу данных как некий набор связанных данных, а систему управления базами данных, или СУБД (Database Management System — DBMS), как программное обеспечение, которое управляет доступом к этой базе данных.

По мере развития информационных технологий совершенствовались методы решения экономических задач на компьютерах. На начальном этапе задачи решались изолированно, а прикладные программы, реализующие их, сами обеспечивали ввод и организацию необходимых данных. Такая технология приводила к значительному дублированию хранимых данных и затрудняла их обновление. Большие объемы экономической информации, ее относительно высокая стабильность, наряду с требованиями к актуальности и достоверности, привели к необходимости интеграции данных в единой базе, обеспечивающей решение всего комплекса задач определенной предметной области. В этой связи потребовалось разработать специальные методы и механизмы управления такого рода совместно используемыми ресурсами данных, которые стали называться базами данных (БД). На концепции БД базируются основные идеи современной информационной технологии. Согласно этой концепции, основой информационной технологии являются данные, которые должны быть организованы в БД с целью адекватного отображения изменяющегося реального мира и удовлетворения информационных потребностей пользователей.

Увеличение объема и структурной сложности хранимых данных, расширение круга пользователей информационных систем выдвинуло требования создания удобных общесистемных средств интеграции хранимых данных и управления ими. Это и привело к появлению в конце 60-х годов первых промышленных систем управления базами данных (СУБД) - специализированных программных средств, предназначенных для организации и ведения БД.

Наряду с разработкой научных основ сформировалась и получила массовое распространение практическая технология баз данных со всеми ее ключевыми компонентами. Созданы методология проектирования и эксплуатации систем баз данных, а также развитые инструментальные средства для разработчиков таких систем и персонала

администратора базы данных, для разнообразных по характеру потребностей и по уровню квалификации категорий пользователей.

1.1. История развития технологий баз данных

Концепция базы данных, зародившаяся около полувека назад, оказалась весьма плодотворной. На ее основе сформировался новый важный пласт информационных технологий, которые эффективно используются для управления данными в экономических информационных системах, научных исследованиях, инженерном деле, образовании, культуре и во многих других областях. Эта концепция положила начало актуальной и активно развивающейся самостоятельной научной дисциплине, специальному разделу информатики.

Широкое практическое использование технологий баз данных стало возможным благодаря созданной за прошедшие годы мощной индустрии реализующего их программного обеспечения, а также активным исследованиям в этой области.

Истоки технологий баз данных относятся к началу 60-х гг., когда был уже накоплен некоторый опыт решения задач обработки экономической информации средствами технологии файловых систем и языка COBOL. Это были технологии, основанные на использовании магнитных лент с их последовательным доступом. Появление таких устройств памяти прямого доступа, как магнитные диски вычислительных систем IBM/360 или ICL-1900, открыло принципиально новые возможности и стимулировало поиски новых эффективных методов организации хранения во внешней памяти интегрированных совокупностей сложно структурированных данных большого объема. Необходимость в них была продиктована потребностями многих разнообразных приложений.

Актуальность и новизна проблем, связанных с этой сферой, привлекла к ним внимание широкого круга специалистов в американских компаниях, занятых производством вычислительной техники и программного обеспечения, а также в крупных индустриальных компаниях, испытывающих наиболее острые потребности в новом инструментарии и ранее других осознавших его эффективность.

В результате активной деятельности ряда крупных компаний США, большинство из которых выполняло военные заказы, стали появляться первые приложения, использующие принципы баз данных. В июне 1963 г. В Санта-Монике (штат Калифорния) компанией System Development Corporation (SDC) был организован, вероятно, первый симпозиум, посвященный проблематике баз данных. На симпозиуме обсуждался ряд докладов по использованию баз данных в военных приложениях, были представлены ранние программные системы, которые можно квалифицировать как СУБД.

По мнению известного эксперта в области баз данных Т. Олле [1], именно в рамках разработок, представленных на этом симпозиуме, и родился термин «база данных».

Начальная стадия развития технологий баз данных. Период 60-ч гг. стал временем становления новых технологий, связанных с созданием, поддержкой и использованием баз данных. В начале 60-х гг. были созданы первые системы управления базами данных. Среди них СУБД общего назначения IDS (Integrated Data Storage, 1963г.), разработанная в компании General Electric под руководством будущего Тьюринговского лауреата Чарльза Бахмана. Эта система интересна не только тем, что она была одной из первых коммерческих СУБД. Реализованные в ней принципы организации базы данных и манипулирования данными стали впоследствии основой сетевой модели данных CODASYL.

В этот период начинают также формироваться основы методологии построения систем баз данных, которая вскоре стала играть основополагающую роль в разработке информационных систем самого различного назначения. Одним из ключевых элементов этой

методологии является концепция модели данных. Термин «модель данных» вошел в лексикон специалистов в области баз данных несколько позднее – в 70-е гг., после публикации фундаментальной работы Эдгара Кодда [2] о реляционной модели данных. Рабочей группе CODASYL по базам данных (CODASYL DBTG), созданной в 1967 г. и преобразованной в последствии (в1971 г.) в Комитет по языку определения данных (Data Definition Language Committee, DDLC), принадлежит заслуга создания спецификаций сетевой модели данных. Эти спецификации стали фактически первым индустриальным стандартом в области систем баз данных. В спецификациях CODASYL DBTG сформулированы такие основополагающие принципы, как: отделение описания данных от прикладной программы и введение концепции схемы базы данных, строгое разграничение «логического» и «физического» представлений данных, обеспечение прозрачности представления в среде хранения для пользователя, концепции защиты целостности данных и управления доступом. Определены функции администрирования данными и принципы построения интерфейсов прикладного программирования СУБД, введена концепция процедуры баз данных – прообраза триггеров в SQL.

Трехуровневая архитектура СУБД, предложенная в спецификациях DBTG, без сомнения, оказала влияние на формирование архитектурной концепции известного отчета ANSI/X3/SPARC [3] – «трехсхемной технологии» - и послужила ее прототипом.

Формировался также иной подход к созданию систем баз данных, в основе которого использовалась иерархическая структуризация данных в базе данных. Этот подход был детально проработан компанией North American Rockwell.

Наряду с разработками средств для «логического» моделирования данных в конце 60-х гг. стали исследоваться методы организации среды хранения базы данных. В подходе CODASYL с самого начала предусматривалось использовать для управления представлением базы данных в среде хранения специальный язык Data Storage Definition Language (DSDL). Работа над этим языком оказалась довольно продолжительной. Его спецификация была одобрена DDLC и опубликована только в 1978 г.

Эффективные методы хранения и доступа для иерархических баз данных разрабатывались в компании IBM в процессе создания ее СУБД IMS. В отличие от большинства других СУБД, где механизмы среды хранения разрабатывались над средствами файловой системы используемой аппаратно-программной платформы, IBM для достижения высокой эффективности доступа ввела специальные методы доступа в саму файловую систему.

Первые шаги индустриального производства СУБД. В конце 60-х гг. начала формироваться и индустрия программного обеспечения систем баз данных. В это время были созданы широко известные коммерческие СУБД общего назначения. Компания IBM разработала свою знаменитую систему IMS (1969 г.), основанную на иерархической модели данных. СУБД IDMS (Integrated Data Management System), созданная компанией BF Goodrich Chemical Company (1971 г.) и ставшая позднее (1973 г.) собственностью Cullinane Corp., была основана на модели данных CODASYL. После вхождения Cullinane Corp. в состав Computer Associates правами на систему IDMS стала обладать эта компания, которая до сих пор продолжает ее поставлять и развивать.

В 1969 г. в рамках Association for Computing Machinery (ACM), крупнейшей международной научно-общественной организации в компьютерной области, было образовано специальное подразделение – ACM SIG on File Description and Translation (ACM SIGFIDET), которое позднее было переименовано в ACM SIG on Management of Data (ACM SIGMOD). Создание SIGFIDET означало, что теория и технологии управления данными получили признание в качестве самостоятельной актуальной ветви информатики и компьютерных технологий.

Основные итоги. 60-е гг. – это период зарождения и успешного становления технологий баз данных, формирования их методологических основ, рождения индустрии

программного обеспечения систем баз данных и организационного оформления сообщества специалистов, работающих в этой области.

Реляционные системы. Период 70-х гг. чрезвычайно богат новыми идеями и подходами в области управления данными, фундаментальными исследованиями, затрагивающими фактически все важнейшие аспекты организации и функционирования систем баз данных, исследовательскими прототипами и коммерческими программными продуктами. В отличие от интуитивно формировавшихся базовых принципов технологий баз данных раннего периода в 70-е гг. происходило интенсивное развитее функциональности СУБД на основе серьезных теоретических исследований, создания прототипов, проведения экспериментов и анализа их результатов. При этом проблематика теоретических исследований была настолько широкой, что можно утверждать – в 70-е гг. были сформированы теоретические основы современных технологий баз данных и образовалась новая самостоятельная научная ветвь информатики – наука о базах данных.

Рождение реляционного подхода к базам данных стало, без сомнения, одним из центральных событий рассматриваемого периода. Публикация статьи Э. Кодда [2] стимулировала целый ряд математических исследований, направленных на изучение свойств реляционных баз данных, что привело, в конечном счете, к созданию теории реляционных баз данных.

Осознавая перспективность реляционного подхода, компания IBM инициировала ряд исследовательских проектов для фундаментального изучения его реализации и создания основ для разработки коммерческих СУБД. Первым из них был получивший широкую известность проект DIAM (Data Independence Access Method) группы М. Сенко. Этот проект позволил изучить возможности моделирования предметной области с помощью предложенной авторами бинарной сетевой модели.

В проекте М. Злуфа, целью которого было создание графического языка запросов, основанного на модели n-арных отношений, разработан язык Query-by-Example (QBE), который сочетает в себе возможности реляционного исчисления с переменными-картежами и реляционного исчисления с переменными на доменах, а также обладает большой выразительной силой. QBE нашел применение в ряде коммерческих реляционных СУБД, где интерфейс QBE сосуществует с поддержкой языка SQL.

Наиболее масштабным из исследовательских работ стал проект System R, выполнявшийся во второй половине 70-х гг. группой под руководством Д. Чамберлина (IBM). Этот проект, направленный на создание функционально развитого прототипа реляционной СУБД, стал одним из главных исследовательских полигонов для комплексного исследования проблем и эффективных методов полномасштабной реляционной СУБД, основанной на n-арной реляционной модели. В его рамках разрабатывались и тщательно исследовались язык пользовательского интерфейса, архитектура системы, организация среды хранения, методы параллельной обработки запросов и механизмы обеспечения целостности данных, подходы к оптимизации запросов, изучались многие другие проблемы. Одним из важных результатов проекта стало создание языка SEQUEL-2 [4], прототипа международного стандарта реляционного языка запросов SQL.

Выполненные в рамках проекта System R исследования обеспечили основу для коммерческих реализаций реляционных СУБД SQL/DS и DB2 компании IBM, а также для реляционных СУБД других поставщиков.

В 70-е гг. довольно многое было сделано в разработке и исследовании различных подходов к моделированию данных, инициированных циклом работ Э. Кодда:

- создание теории зависимостей и базирующейся на ней теории нормализации отношений, которая стала основой проектирования реляционных баз данных;
- разработка алгоритмов редукции выражений реляционного исчисления в реляционную алгебру;

- первые шаги в исследовании неопределенных значений и проблемы неполноты информации;
- ряд результатов, связанных с понятием универсального отношения.

Большой цикл работ разных авторов был выполнен с целью повышения семантического уровня моделей данных. Предложенная П.Ченом модель «сущностей-связей» породила целое направление исследований и ряд технологий, связанных с концептуальным проектированием баз данных. Она до сих пор является основой ряда инструментальных средств CASE. С 1979 г. стала регулярно проводиться международная научно-техническая конференция, посвященная исследованиям различных аспектов этой модели и ее применениям.

Создание основ техники хранения данных. Методов доступа, доставшихся формирующимся технологиям баз данных в наследство от техники файловых систем, оказалось явно недостаточно для эффективного управления данными в среде хранения баз данных. Поэтому в рассматриваемый период уделялось много внимания развитию новых методов доступа. В ряде СУБД стали использовать страничную организацию пространства памяти с индексами страниц, обеспечивающими косвенную адресацию хранимых записей, благодаря чему достигалась подвижность записей на странице. В базах данных нашли также применение методы хеширования.

Развитие в середине 70-х гг. исследований и разработок, связанных с проектированием систем баз данных, привело к осознанию необходимости технологий и инструментария управления метаданными в крупных системах. Инструментальные средства такого рода, ориентированные на разработчика, системный персонал и пользователей таких систем, стали называть словарями данных (Data Dictionary). Средства, предназначенные для поддержки функционирования самой системы, назывались справочниками данных (Data Directory). Далее средства обоих типов стали объединять в единую систему словаря/справочника данных (Data Dictionary/Directory, DD/D).

За прошедшие десятилетия развития систем баз данных и информационных систем были созданы и нашли применение в этой области разнообразные и сложным образом взаимосвязанные информационные технологии, призванные решать многочисленные возникающие здесь задачи. При создании таких систем применяются не только технологии, специально разработанные для систем баз данных и управления информационными ресурсами в традиционном их понимании. Поэтому технологии объектного анализа и проектирования систем программного обеспечения имеют сегодня прямое отношение к объектным системам баз данных и информационным системам. Успешные информационные технологии начали широко распространяться, реализовываться многими поставщиками на различных аппаратно-программных платформах, интегрироваться с другими технологиями в рамках одного приложения.

Все это породило новые проблемы, такие, как обеспечение переносимости приложений между различными программно-аппаратными платформами, интероперабельности программных продуктов различных поставщиков и созданных на их основе приложений, повторного использования ресурсов, в частности метаданных и программных компонентов приложений, измерения производительности различных систем с возможностью сопоставления результатов измерений и многие другие.

1.2. Процессы стандартизации в области технологий баз данных

В связи с большими масштабами проводимых в настоящее время разработок информационных систем всевозможного назначения наибольшую актуальность приобрели технологические проблемы обеспечения интероперабельности (интероперабельность

означает упрощение комплексирования новых программных систем на основе использования готовых компонентов со стандартными интерфейсами) программных средств разных поставщиков, переносимости ресурсов данных и приложений между различными средами, возможностей повторного использования и интеграции программных и информационных ресурсов.

Необходимым условием решения указанных задач является разработка стандартов технологий и отдельных их элементов. Сложность и разветвленность технологий баз данных, многообразие сфер их применения и их специфических особенностей способствовали вовлечению в деятельность по стандартизации довольно широкого круга организаций – от официальных национальных и международных органов стандартизации до различных специально учрежденных индустриальных консорциумов. Их усилиями в рассматриваемой области созданы, продолжают поддерживаться развиваться многочисленные стандарты, связанные с разными аспектами технологий баз данных и их применений.

Такие стандарты представляют собой спецификации необходимых свойств системной архитектуры (функциональности, интерфейсов и др.), протоколов, процессов, моделей, языковых средств представления программного кода, запросов, описания данных или метаданных, методов и/или средств эталонного тестирования программных продуктов, а также используемой терминологии. В связи с многообразием объектов стандартизации в рассматриваемой области, требований к ним, используемых подходов, существованием многочисленных консорциумов, групп, институтов и других организаций, занимающихся стандартизацией, создано огромное количество разнообразных стандартов.

Среди существующих стандартов различаются стандарты де-юре и стандарты де-факто. Стандарты де-юре разрабатываются и принимаются (или, как иногда говорят, публикуются) официальными организациями, специально учрежденными для деятельности в этой области. Поэтому стандарты де-юре называют также официальными стандартами.

Функционируют национальные и международные официальные организации по стандартизации. Принятые ими стандарты имеют соответственно статус национальных или международных. Среди международных официальных организаций, учреждающих стандарты, главной является Международная организация по стандартизации (International Organization for Standardization, ISO). Другой крупной авторитетной международной организацией, занимающейся стандартизацией в области информационных технологий, является Международная электротехническая комиссия (International Electro technical Commission, IEC). Для обеспечения взаимодействия по разработке стандартов информационных технологий ISO и IEC образовали Совместный технический комитет (Joint Technical Committee 1, JTC1). Его усилиями к настоящему времени уже разработано значительное количество совместных стандартов ISO/IEC.

Среди национальных организаций по стандартизации наибольший вклад в создание стандартов в области информационных технологий внес и продолжает вносить Американский национальный институт стандартов (American National Standards Institute, ANSI). ANSI не занимается непосредственной разработкой стандартов. Это делают аккредитованные при нем организации, которые разрабатывают проекты стандартов и представляют их в ANSI для придания им официального статуса, для чего представленные проекты подвергаются установленной в ANSI процедуре.

Стандартами в области технологий баз данных занимался комитет X3H2. В 1997 г. группа X3 была реорганизована в Национальный комитет по стандартам информационных технологий (National Committee for Information Technology Standards, NCITS). Технический комитет H2 продолжает функционировать в NCITS, сохраняя свое название. Силами этого комитета был подготовлен, в частности, стандарт удаленного доступа к базам данных(Remote Database Access, RDA), ведется работа по развитию стандарта языка SQL, по мультимедийным и прикладным пакетам SQL (SQL Multimedia and Application Packages, SQL/MM).

Наряду со стандартами де-юре существует также категория стандартов де-факто. Это такие стандарты, которые широко применяются на практике, независимо от наличия или отсутствия у них официального статуса.

Стандарты де-факто могут создаваться самыми различными организациями. Сегодня это спецификации, предложенные не только ISO или ANSI, но и многочисленными индустриальными консорциумами, большинство из которых было учреждено в 90-е гг., или реже отдельной компанией. Ряд наиболее авторитетных консорциумов, занимающихся стандартизацией в рассмотренной здесь области (ODMG,OMG,MDC,TPC,W3C и др.) Наиболее распространенными стандартами де-факто в области баз данных и информационных технологий являются, в частности, стандарт ANSI/ISO/IEC SQL-92, стандарты объектных данных консорциума ODMG, разработанные OMG стандарты CORBA и UML, принятый W3C стандарт расширяемого языка разметки XML.

ISO и некоторые другие организации, уполномоченные принимать стандарты де-юре, учредили сокращенную процедуру (Fast Track Process) рассмотрения общедоступных и широко признанных спецификаций претендующих на статус официальных стандартов. Такие спецификации называются доступными (Public Available Specifications),а процедура их рассмотрения включает единственный цикл изучения и голосования (Review-and-Vote). По такой процедуре в настоящее время рассматриваются некоторые стандарты ANSI и Open Group с целью придания им статуса стандартов ISO.

Рассматривая функциональную направленность стандартов информационных технологий, необходимо выделить важнейшую их группу – стандарты открытых систем [6]. Стандарты этой категории обеспечивают переносимость программного обеспечения и информационных ресурсов с одной платформы на другую, тот или иной уровень интероперабельности программных продуктов различных поставщиков и информационных ресурсов.

По сферам применения различаются стандарты, которые не зависят от области применения, и стандарты, предназначенные для конкретной предметной области. Первые из них называют стандартами «горизонтального рынка», а вторые – стандартами «вертикальных рынков».

При разработке стандартов часто используется подход, предусматривающий конкретизацию или уточнение некоторых элементов данного стандарта с учетом специфических потребностей каких-либо особых случаев применения. Такая специализация стандарта называется его профилем для рассматриваемой категории применений. Таким образом, профиль обеспечивает расширение и обогащение функциональности стандарта для некоторой конкретной категории применений. Под профилем иногда понимается также совокупность совместимых стандартов с различной функциональностью, предназначенная для обеспечения потребностей некоторой сферы применения.

В области стандартизации технологий баз данных и информационных систем наряду с созданием необходимых стандартов важнейшее значение имеет и проблема сертификации разрабатываемых программных продуктов на соответствие тем или иным стандартам. Большую работу проводит консорциум Open Group по сертификации программных средств на соответствие стандартам открытых систем.

Первой успешной попыткой считается индустриальный стандарт CODASYL DBTG, начальная версия которого была опубликована в 1969 году. Выпущенная в 1971 г. версия этого стандарта получила широкое признание и стала основой разработки целого ряда коммерческих СУБД. Одной из первых систем этого семейства является СУБД IDMS, установки поздних версий которой, поставляемых и развиваемых теперь компанией Computer Associates, эксплуатируются до настоящего времени.

Осознание той важной роли, которую стали играть технологии баз данных в разработках информационных систем, и интенсивное расширение масштабов этих разработок побудили также и официальные органы стандартизации обратить внимание на

эту сферу деятельности. Одной из первых попыток в этом направлении, предпринятых официальными органами стандартизации, принадлежит исследовательской группе по базам данных ANSI/X3/SPARC. Эта группа стремилась выявить возможные субъекты стандартизации в технологиях баз данных. Результаты проведенной ею работы были представлены в знаменитом отчете [7], который дал путевку в жизнь концепциям трехсхемной технологии и способствовал широкому признанию принципов многоуровневого представления данных в системах баз данных.

Следующим заметным шагом в области стандартизаций технологий баз данных стало создание в 1977 г. под влиянием указанного отчета ANSI/X3/SPARC рабочей группы ISO, на которую возлагалась задача выработки концепций и терминологии концептуального моделирования предметной области в системах баз данных, анализа и выбора возможных претендентов на роль инструментального средства для этих целей. Результаты деятельности этой группы не завершились созданием какого-либо конкретного стандарта, однако значительный интерес вызвал подготовленный ею отчет [8].

Первым официальным стандартом в области технологий баз данных был принятый ANSI стандарт языка SQL (1986 г.), получивший статус национального стандарта США. Первым международным стандартом в рассматриваемой области стал стандарт языка SQL, одобренный ISO/IEC в начале 1987 г. Эти стандарты довольно скоро стали и стандартами де-факто.

В 1987 г. был принят в качестве стандарта ISO/IEC язык определения данных сетевой модели данных Network Database Language (NDL), разработанный на основе языка определения данных CODASYL.

Относительно доступности документации по стандартам нужно отметить, что в отличие от многих индустриальных консорциумов, предоставляющих электронные версии спецификаций разработанных ими стандартов безвозмездно на их Web-сайтах, органы официальной стандартизации (ISO, ANSI и др.) распространяют их только на коммерческой основе.

Перед разработчиками крупной системы встает весьма ответственная задача выбора среди возможных альтернатив и определения совместимой совокупности стандартов, на которой будет основана предпринимаемая разработка. Задача эта многокритериальная, и должны учитываться экономические, технологические и другие критерии.

Компетентный руководитель проекта, принимая решение, обычно исходит не только из имеющихся ограничений (сроки, финансовые ресурсы, имеющийся персонал и его уровень квалификации и т.д.), но учитывает и такие факторы, как функциональные возможности определяемой стандартом технологии, степень открытости стандарта, обеспечение интероперабельности с другими технологиями, степень популярности и перспективы данного стандарта, прогнозируемый срок его жизни.

1.2.1. Стандарт языка SQL

На рубеже 70-х гг. наряду с активным развитием доминировавших в этот период графовых моделей данных (сетевой модели CODASYL, иерархической модели, модели связанных файлов и т. д.) и выпуском целого ряда поддерживающих их коммерческих СУБД родился новый подход к моделированию данных, основанный на математической теории отношений.

Фундаментом для них послужил цикл новаторских работ Э. Кодда, и прежде всего знаменитая его статья [2], в которых впервые были представлены математической основы реляционной модели данных, разработаны основы реляционной алгебры и реляционного исчисления в реляционную алгебру (алгоритм редукции), создан один из первых

реляционных языков Alpha, ставший эталоном реляционной полноты других языков – концепции, введенной Э. Коддом для оценки функциональности реляционных языков, разработаны основы теории нормализации отношений, на которой базируются методологии проектирования реляционных баз данных.

Одним из этапов этих исследований являлась разработка языка высокого уровня, воплощающего функциональность реляционной модели, который был бы не только достаточно развит в функциональном отношении, но и допускал бы эффективную реализацию. Исследовались различные подходы – бинарные и n-арные отношения, алгебраические языки и языки исчисления и т.п. В результате этих работ был создан целый ряд реляционных языков – Alpha, ISBL, SQUARE [9], Query-by-Example, FORAL и, наконец, ранняя версия SQL – язык SEQUEL [4].

Значительное влияние на развитие реляционных языков и методов реализации реляционных систем оказала осуществляемая в тот же период в Калифорнийском университете (Беркли) группой М. Стоунбеккера разработка экспериментальной реляционной СУБД INGRES.

В проекте System R языку запросов уделялось серьезное внимание вплоть до многоаспектного тестирования его пользователей, изучение статистики ошибок и внесения соответствующих изменений в синтаксические спецификации. В результате такой тщательной доработки языка в 1976-1977 гг. была создана его пересмотренная версия SEQUEL-2 [4], которую авторы впоследствии и стали называть для краткости SQL.

Хотя компании IBM принадлежат лавры первооткрывателя реляционной модели данных и она внесла огромный конструктивный вклад в разработку теории реляционных баз данных и в создание технологий реализации реляционных СУБД, автором первой коммерческой реляционной СУБД, поддерживающей язык SQL, тем не менее стала другая компания. В 1979 г. компанией Relation Software Inc., которая впоследствии стала называться Oracle Corporation, была выпущена первая реляционная коммерческая СУБД, базирующаяся на языке SQL.

В 1986 г. ANSI одобрил в качестве национального стандарта языка реляционных систем баз данных спецификации версии языка SQL, подготовленные комитетом X3H2 (теперь это комитет H2 NCITS). Принятый ANSI стандарт SQL-86 был вскоре (1987 г.) практически без каких-либо изменений одобрен ISO и стал первым международным стандартом языка SQL.

Проект SQL2 завершился в 1992 г. принятием следующей версии стандарта ISO/IEC и ANSI – SQL-92 [29]. В нее были включены многие отсутствующие в SQL-89 функциональные возможности. Среди них – динамический SQL и спецификации встроенного SQL для ряда популярных включающих языков программирования (Ada, C, COBOL, FORTRAN, MUMPS, Pascal и PL/1), новые типы данных, дополнительные возможности определения данных, средства манипулирования схемой, управления привилегиями доступа и декларации ограничений целостности данных. Усовершенствована техника работы с курсорами, введены также и некоторые другие возможности.

В 1990 г. возник проект SQL3. Предусматривалось, в частности включение в SQL3 объектных возможностей, типов данных, определяемых пользователем, поддержка триггеров, временных (темпоральных) свойств данных, средств для управления внешними и мультимедийными данными.

С учетом неотложных потребностей индустрии программного обеспечения реляционных систем баз данных, не ожидая окончания работы над SQL3, были стандартизованы некоторые функциональные возможности языка, которым решено было придать статус расширенной SQL-92. К их числу относятся стандарт интерфейса уровня вызовов SQL/CLI, так называемый новый стиль связывания для SQL, а также стандарт хранимых процедур SQL/PSM.

Еще один компонент будущего нового стандарта SQL родился в связи с быстрым ростом популярности языка Java и стремлением использовать его в качестве инструмента разработки приложений систем баз данных. Образованный в 1997 г. консорциум SQLJ разработал спецификации связывания SQL для языка Java, интерфейсов высокого уровня, которые обеспечивают взаимодействие Java-программ с системами баз данных. Часть 0 спецификаций SQLJ была одобрена ANSI в 1998 г. и включена как один из компонентов в проект будущего нового стандарта SQL. В 1999 г. ANSI была одобрена часть 1 SQLJ. Работа над официальной стандартизацией части 2 SQLJ завершилась ее одобрением в ноябре 2000 г. [9 - 12].

Актуальность стандарта SQLJ в настоящее время довольно велика и в связи с ростом числа установок объектно-реляционных серверов баз данных, поставляемых ведущими поставщиками СУБД. Уже имеются его коммерческие реализации.

Рассматривая инфраструктуру стандарта SQL, следует упомянуть также международный стандарт удаленного доступа к SQL-базам данных, новая версия которого, разработанная JTC1, была одобрена ISO/IEC [13].

В настоящее время стандарт языка SQL является, пожалуй, наиболее широко используемым стандартом в области технологий баз данных.

1.*2.2. ODMG и стандарты объектных баз данных*

Индустриальный консорциум Object Data Management Croup играет определяющую роль в разработке стандартов объектных баз данных. Важнейшие стандарты объектных баз данных сегодня – это стандарты консорциума ODMG.

Активные исследования в области языков программирования в конце 60-х – начале 70-х гг., связанные с типизацией данных, привели к формированию концепции абстрактного типа данных, которая нашла конструктивное воплощение в разработанных в это время исследовательских языках программирования – CLU, Alphard и др. [14, 15]. Обогащенная идеями наследования, полиморфизма и инкапсуляции реализации, эта концепция трансформировалась впоследствии в подход, который и стал называться объектным. Получили распространение созданный еще в 1972 г. в компании Xerox PARC объектный язык Smalltalk, объектное расширение популярного языка C, язык C++ (опубликованный в 1983 г. Б. Строустрапом в AT&T Bell Laboratories), и системы программирования, реализующие этот язык на платформе ПК, программирование в объектной парадигме стало массовым явлением.

Эта ситуация в значительной мере пробудила интерес к воплощению объектной парадигмы в технологиях баз данных. Проблема состояла прежде всего в том, что возникла необходимость в обеспечении эффективной среды хранения объектных данных для поддержки многочисленных прикладных программных систем, реализованных средствами объектных языков. Эту функцию было естественно возложить на СУБД, основанные на объектных моделях данных и обладающие интерфейсами прикладного программирования (API) для объектных языков.

Развитие производства ряда коммерческих объектно-ориентированных СУБД выдвинуло в начале 90-х гг. вопрос о необходимости стандартизации модели данных и языковых средств для таких систем с тем, чтобы обеспечить переносимость приложений между средами различных объектных СУБД, а также их интероперабельность. Именно с целью организации практической деятельности по созданию стандартов для объектных систем баз данных в июне 1991 г. был учрежден индустриальный консорциум Object Database Management Group (ODMG), который объединил всех ведущих поставщиков объектных СУБД.

В качестве исходной платформы для разработки своего стандарта объектных СУБД консорциум ODMG использовал:

- манифест систем объектно-ориентированных баз данных [16], в котором были определены основные свойства ООСУБД;
- объектную модель и язык определения интерфейсов (IDL) стандарта OMG CORBA 1.0, обеспечивающего создание распределенных неоднородных интероперабельных объектных сред;
- проект стандарта SQL2 (в этот период завершились работы над SQL-92 [9]).

Главные требования к разрабатываемому ODMG стандарту заключались в обеспечении переносимости приложений между средами различных ООСУБД и интероперабельности с архитектурой CORBA. В августе 1993 г. консорциум опубликовал первую версию стандарта – ODMG-93 Release 1.0 [17]. В ней содержались спецификации объектной модели, языка определения объектов, объектного языка запросов и связывания для объектных языков программирования Smalltalk и C++. При этом объектная модель ODMG представляла собой расширение модели-ядра стандарта OMG CORBA, а язык определения объектов ODL – соответственно расширение языка определения интерфейсов OMG IDL. Благодаря этому обеспечивалась естественная возможность интеграции систем баз данных, основанных на стандарте ODMG, в среду объектных технологий CORBA. Язык запросов OQL в стандарте явился результатом адаптации некоторого фрагмента языка SQL к объектной модели ODMG.

С их появлением спецификации ODMG-93 приобрели статус стандарта де-факто, поскольку в их создании участвовали все ведущие производители объектных СУБД, которые поддерживали этот стандарт собственными программными продуктами.

Стандарт ODMG-93. ODMG-93 – это первый разработанный консорциумом стандарт для объектных СУБД [16]. Он был опубликован в 1994 г. издательством Morgan Kaufmann Publishers (США), которое впоследствии стало постоянным издателем спецификаций стандартов ODMG.

Главные требования, которые определяли характер этого стандарта, заключались в обеспечении переносимости приложений относительно различных ООСУБД, а также в интероперабельности с архитектурой CORBA. По замыслу авторов стандарт должен был интегрировать концепции объектной модели OMG, языка баз данных SQL и объектно-ориентированных языков программирования, таких, как C++, и опираться на результаты деятельности по стандартизации в каждом из этих направлений.

Основными компонентами стандарта ODMG-93 являются:

- объектная модель, которую, как предполагается, должны поддерживать ООСУБД;
- опирающийся на эту модель язык определения объектов ODL;
- основанный на этой модели и на языке SQL объектный язык запросов OQL;
- языки манипулирования данными (Object Manipulation Language – OML) для различных включающих языков, представляющие собой, по существу, отображения объектной модели ODMG в объектно-ориентированные языки программирования Smalltalk и C++ (эти отображения называют часто связываниями для соответствующих языков программирования).

Стандарт ODMG 2.0. Эта версия стандарта, названного стандартом объектных баз данных, была принята ODMG в июле 1997 г. [18]. Она представляет собой существенным образом пересмотренные, доработанные и расширенные спецификации версии ODMG-93. В спецификации стандарта ODMG 2.0 включено новое, уточненное и расширенное описание объектной модели.

В стандарте ODMG 2.0 радикальным образом, но, к сожалению, не безупречно переработана объектная модель. Внесены соответствующие поправки в синтаксис языков ODL и OQL. Стандарт ODMG 2.0 расширил спецификации ODMG:

- дополнительной функциональностью базовой объектной модели стандарта;
- введением репозитория метаданных ODL-схемы;
- спецификациями связывания для языка Java;
- стандартизацией внешней формы как данных, так и метаданных для дампа базы данных.

Стандарт ODMG 3.0. В январе 2000 г. консорциум ODMG опубликовал новую версию своего стандарта – ODMG 3.0 [19], который называется стандартом объектных данных. В ODMG 3.0 пересмотрены и уточнены спецификации объектной модели, введен ряд улучшений в связывание для языка Java, которое уже реализовано в нескольких программных продуктах.

1.3. Трехуровневая архитектура ANSI/SPARC

Одно из преимуществ подхода, основанного на применении баз данных, - абстрагирование данных (data abstraction), заключается в том, что можно изменить внутреннее определение объекта без каких-либо последствий для его пользователей, при условии, что внешнее определение объекта остается неизменным.

В базе данных структура данных отделена от приложений и хранится в базе данных. Добавление новых структур данных или изменение существующих никак не влияет на приложения, при условии, что они не зависят непосредственно от изменяемых компонентов. Например, добавление нового поля в запись или создание нового файла никак не повлияет на работу имеющихся приложений. Однако удаление поля из используемого приложением файла повлияет на это приложение, а потому его также потребуется соответствующим образом модифицировать.

Поскольку база данных является общим ресурсом, то каждому пользователю может потребоваться свое, отличное от других представление о характеристиках информации, сохраняемой в базе данных. Для удовлетворения этих потребностей архитектура большинства современных коммерческих СУБД в той или иной степени строится на базе так называемой архитектуры ANSI/SPARC.

Первая попытка создания стандартной терминологии и общей архитектуры СУБД была предпринята в 1971 году группой, называемой DBTG. Она была создана, после конференции CODASYL (Conference on Data Systems and Languages — Конференция по языкам и системам данных), прошедшей в этом же году.

Комитет ANSI/SPARC признал необходимость использования трехуровневого подхода, т.е. необходимость воплощения независимого уровня для изоляции программ от особенностей представления данных на более низком уровне (Guide/Share,1970). Хотя модель ANSI/SPARC не стала стандартом, тем не менее, она представляет собой основу для понимания некоторых функциональных особенностей СУБД.

Для нас наиболее важным является идентификация трех уровней абстракции, т.е. трех различных уровней описания элементов данных. Эти уровни формируют трехуровневую архитектуру, которая охватывает внешний, концептуальный и внутренний уровни, как показано на рис.1.1.

Цель трехуровневой архитектуры заключается в отделении пользовательского представления базы данных от ее физического представления. Ниже перечислено несколько причин, по которым желательно выполнять такое разделение.

- Каждый пользователь должен иметь возможность обращаться к одним и тем же данным, используя свое собственное представление о них. Каждый пользователь

должен иметь возможность изменять свое представление о данных, причем это изменение не должно оказывать влияния на других пользователей.

- Пользователи не должны непосредственно иметь дело с такими подробностями физического хранения данных в базе, как индексирование и хеширование. Иначе говоря, взаимодействие пользователя с базой не должно зависеть от особенностей хранения в ней данных.

Рис.1.1. Трехуровневая архитектура описания данных ANSI/SPARC

- Администратор базы данных (АБД) должен иметь возможность изменять структуру хранения данных в базе, не оказывая влияния на пользовательские представления.
- Внутренняя структура базы данных не должна зависеть от таких изменений физических аспектов хранения информации, как переключение на новое устройство хранения.
- АБД должен иметь возможность изменять концептуальную или глобальную структуру базы данных без какого-либо влияния на всех пользователей.

Уровень, на котором воспринимают данные пользователи, называется внешним уровнем (external level), тогда как СУБД и операционная система воспринимают данные на внутреннем уровне (internal level). Именно на внутреннем уровне данные реально сохраняются с использованием всех тех структур и файловой организации, которые рассматриваются. Концептуальный уровень (conceptual level) представления данных предназначен для отображения внешнего уровня на внутренний и обеспечения необходимой независимости друг от друга.

Внешний уровень - представление базы данных с точки зрения пользователей. Этот уровень описывает ту часть базы данных, которая относится к каждому пользователю. Внешний уровень состоит из нескольких различных внешних представлений базы

данных. Каждый пользователь имеет дело с представлением "реального мира", выраженным в наиболее удобной для него форме. Внешнее представление содержит только те сущности, атрибуты и связи "реального мира", которые интересны пользователю. Другие сущности, атрибуты или связи, которые ему неинтересны, также могут быть представлены в базе данных, но пользователь может даже не подозревать об их существовании. Помимо этого, различные представления могут по-разному отображать одни и те же данные. Например, один пользователь может просматривать даты в формате (день, месяц, год), а другой — в формате (год, месяц, день). Некоторые представления могут включать производные или вычисляемые данные, которые не хранятся в базе данных как таковые, а создаются по мере надобности.

Концептуальный уровень - обобщающее представление базы данных. Этот уровень описывает то, какие данные хранятся в базе данных, а также связи, существующие между ними. Промежуточным уровнем в трехуровневой архитектуре является концептуальный уровень. Этот уровень содержит логическую структуру всей базы данных (с точки зрения АБД). Фактически, это полное представление требований к данным со стороны организации, которое не зависит от любых соображений относительно способа их хранения. На концептуальном уровне представлены следующие компоненты:

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

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

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

- распределение дискового пространства для хранения данных и индексов;
- описание подробностей сохранения записей (с указанием реальных размеров сохраняемых элементов данных);
- сведения о размещении записей;
- сведения о сжатии данных и выбранных методах их шифрования.

Ниже внутреннего уровня находится физический уровень (physical level), который контролируется операционной системой, но под руководством СУБД. Однако функции СУБД и операционной системы на физическом уровне не вполне четко разделены и могут варьироваться от системы к системе. В одних СУБД используются многие

предусмотренные в данной операционной системе методы доступа, тогда как в других применяются только самые основные и реализована собственная файловая организация. Физический уровень доступа к данным ниже СУБД состоит только из известных, операционной системе элементов (например, указателей, как реализовано последовательное распределение и хранятся ли поля внутренних записей на диске в виде непрерывной последовательности байтов).

Общее описание базы данных называется схемой базы данных. Существует три различных типа схем базы данных, которые определяются в соответствии с уровнями абстракции трехуровневой архитектуры. На самом высоком уровне имеется несколько внешних схем или подсхем, которые соответствуют разным представлениям данных. На концептуальном уровне описание базы данных называют концептуальной схемой, а на самом низком уровне абстракции — внутренней схемой.

Концептуальная схема описывает все элементы данных и связи между ними, с указанием необходимых ограничений поддержки целостности данных. Для каждой базы данных имеется только одна концептуальная схема.

На нижнем уровне находится внутренняя схема, которая является полным описанием внутренней модели данных. Она содержит определения хранимых записей, методы представления, описания полей данных, сведения об индексах и выбранных схемах хеширования. Для каждой базы данных существует только одна внутренняя схема.

СУБД отвечает за установление соответствия между этими тремя типами схем, а также за проверку их непротиворечивости. Иначе говоря, СУБД должна убедиться в том, что каждую внешнюю схему можно вывести на основе концептуальной схемы. Для установления соответствия между любыми внешней и внутренней схемами СУБД должна использовать информацию из концептуальной схемы.

Концептуальная схема связана с внутренней схемой посредством концептуально внутреннего отображения. Оно позволяет СУБД найти фактическую запись или набор записей на фактическом устройстве хранения, которые образуют логическую запись в концептуальной схеме, с учетом любых ограничений, установленных для выполняемых над данной логической записью операций. Оно также позволяет обнаружить любые различия в именах объектов, именам атрибутов, порядка следования атрибутов, их типах данных и т.д.

Наконец, каждая внешняя схема связана с концептуальной схемой с помощью внешне концептуального отображения. С его помощью СУБД может отображать имена пользовательского представления на соответствующую часть концептуальной схемы.

Важно различать описание базы данных и саму базу данных. Описанием базы данных является схема базы данных. Схема создается в процессе проектировании базы данных, причем предполагается, что она изменяется достаточно редко. Однако содержащаяся в базе данных информация может меняться часто — например, всякий раз при вставке сведений о новом сотруднике или новом объекте сдаваемой в аренду недвижимости. Совокупность информации, хранящейся в базе данных в любой определенный момент времени, называется состоянием базы данных. Следовательно, одной и той же схеме базы данных может соответствовать множество ее различных состояний. Схема базы данных иногда называется содержанием базы данных, а ее состояние — детализацией.

Основным назначением трехуровневой архитектуры является обеспечение независимости от данных, которая означает, что изменения на нижних уровнях никак не влияют на верхние уровни. Различают два типа независимости от данных: логическую и физическую.

Логическая независимость от данных означает полную защищенность внешних схем от изменений, вносимых в концептуальную схему. Такие изменения концептуальной, схемы, как добавление или удаление новых сущностей, атрибутов или связей, должны осуществляться без необходимости внесения изменений в уже существующие внешние схемы или переписывания прикладных программ. Ясно, что пользователи, для которых эти изменения предназначались, должны знать о них, но очень важно, чтобы другие пользователи даже не подозревали об этом.

Физическая независимость от данных означает защищенность концептуальной схемы от изменений, вносимых во внутреннюю схему. Такие изменения внутренней схемы, как использование различных файловых систем или структур хранения, разных устройств хранения, модификация индексов или хеширование, должны осуществляться без необходимости внесения изменений в концептуальную или внешнюю схемы. Пользователем могут быть замечены изменения только в общей производительности системы. На самом деле наиболее распространенной причиной внесения изменений во внутреннюю схему является именно недостаточная производительность выполнения операций.

1.4. Жизненный цикл баз данных

Информационная система – это ресурсы, которые позволяют выполнять сбор, корректировку и распространение информации внутри организации.

Типичная компьютеризированная информационная система включает такие компоненты, как:

- база данных;
- программное обеспечение базы данных;
- прикладное программное обеспечение;
- аппаратное обеспечение, в том числе устройства хранения;
- персонал, использующий и разрабатывающий эту систему.

База данных является фундаментальным компонентом информационной системы, а ее разработку и использование следует рассматривать с точки зрения самых широких требований организации. Следовательно, жизненный цикл информационной системы неотъемлемым образом связан с жизненным циклом системы базы данных, поддерживающей его. Жизненный цикл информационной системы обычно состоит из нескольких этапов:

- планирование,
- сбор и анализ требований,
- проектирование (включая проектирование базы данных),
- создание прототипа,
- реализация,
- тестирование,
- преобразование данных,
- сопровождение.

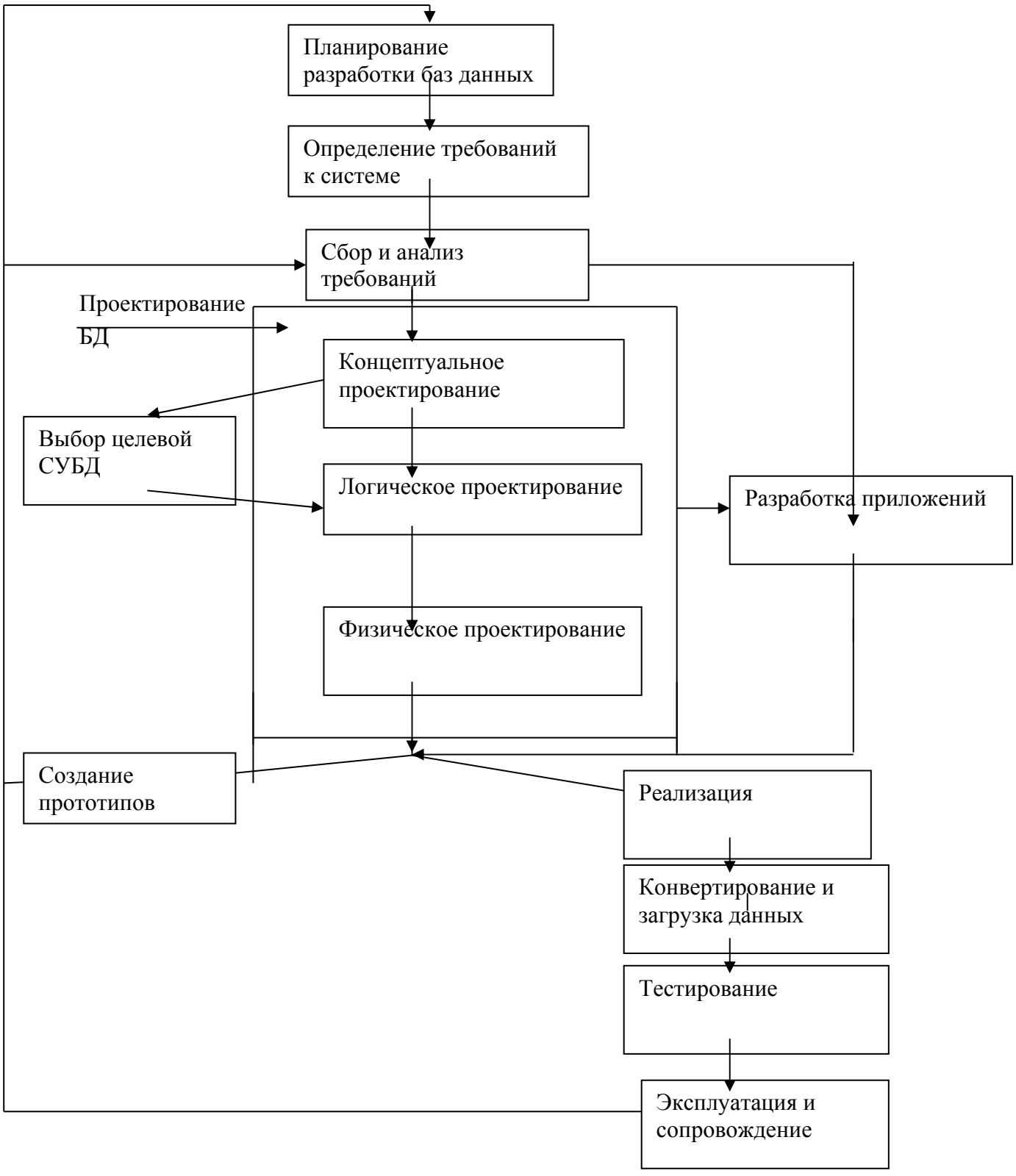

Рис.1.2. Этапы жизненного цикла базы данных

Мы будем рассматривать все этапы жизненного цикла баз данных с более широкой точки зрения — как этапы ЖЦ определенного компонента всей информационной системы в целом. Этапы жизненного цикла базы данных показаны на рис.1.2.

Следует признать, что эти этапы не являются строго последовательными, а включают некоторое количество повторов предыдущих шагов в виде циклов обратной связи (feedback

loop). Например, при проектировании базы данных могут возникнуть проблемы, для разрешения которых потребуется вернуться к этапу сбора и анализа требований. Циклы обратной связи могут возникать почти между всеми этапами, но здесь показаны только наиболее очевидные из них.

Основные действия, выполняемые на каждом этапе жизненного цикла базы данных.

- Планирование разработки базы данных. Планирование самого эффективного способа реализации этапов жизненного цикла системы.
- Определение требований к системе. Определение диапазона действия и границ приложения базы данных, состава его пользователей и областей применения.
- Сбор и анализ требований пользователей. На этом этапе выполняется сбор и анализ требований пользователей из всех возможных областей применения.
- Проектирование базы данных. Полный цикл разработки включает концептуальное, логическое и физическое проектирование базы данных.
- Выбор целевой СУБД (необязательно). На этом этапе выполняется выбор наиболее подходящей СУБД для приложения базы данных.
- Разработка приложений. Определение пользовательского интерфейса и прикладных программ, которые используют и обрабатывают базу данных.
- Создание прототипов (необязательна). На этом этапе создается рабочая модель приложения базы данных, которая позволяет разработчикам или пользователям представить и оценить окончательный вид и способы функционирования системы.
- Реализация. Этот этап включает создание внешнего, концептуального и внутреннего определений базы данных и прикладных программ.
- Конвертцрование и загрузка данных. На этом этапе выполняется преобразование и загрузка данных (и прикладных программ) из старой системы в новую.
- Тестирование. Приложение базы данных тестируется с целью обнаружения ошибок, а также его проверки на соответствие всем требованиям, выдвинутым пользователями.
- Эксплуатация и сопровождение. На этом этапе приложение базы данных считается полностью разработанным и реализованным.

1.5. Организация баз данных

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. Под предметной областью принято понимать часть реального мира, подлежащего изучению. При создании информационной системы требуется как согласование задач (функций), так и согласование данных. При этом были сформулированы основные требования к организации данных:

- интеграция данных;

- максимально возможная независимость данных от прикладных программ.

Выполнение этих требований привело к созданию единого для всех задач блока данных – базы данных и разработке одной управляющей программы для манипулирования данными на физическом уровне – системе управления базой данных.

База данных – это совокупность данных, обладающих следующими свойствами:

- интегрированность, направленная на решение общих задач;

- модельность (структурированность, отражающая некоторую часть реального мира);

- независимость описания данных от прикладных программ.

База данных – это совместно используемый набор логически связанных данных (и описание этих данных), предназначенный для удовлетворения информационных потребностей пользователей.

База данных - это, по сути, единая совокупность данных, которая однократно определяется, а затем используется одновременно многими пользователями. Вместо разрозненных файлов с избыточными данными, здесь все данные собраны вместе с минимальной долей избыточности. База данных уже является общим корпоративным ресурсом. Причем база данных хранит не только рабочие данные, но и их описание. По этой причине базу данных называют набором интегрированных записей с самоописанием. В совокупности описание данных называется системным каталогом (system catalog), или словарем данных (data dictionary), а сами элементы описания принято называть метаданными (meta-data),т.е.”данными о данных”. Именно наличие самоописания данных в базе данных обеспечивает в ней независимость между программами и данными (program- data independence). Подход, основанный на применении базы данных, где определение данных отделено от приложений, очень похож на подход, используемый при разработке современного программного обеспечения, когда наряду с внутренним определением объекта существует его внешнее определение. Пользователи объекта видят только его внешнее определение и не заботятся о том, как он определяется и как функционирует. Одно из преимуществ такого подхода, а именно абстрагирования данных (data abstraction), заключается в том, что можно изменить внутреннее определение объекта без каких-либо последствий для его пользователей, при условии, что внешнее определение объекта остается неизменным. Аналогичным образом, в подходе с использованием баз данных, структура данных отделена от приложений и хранится в базе данных. Добавление новых структур данных или изменение существующих никак не влияет на приложения, при условии, что они не зависят непосредственно от изменяемых компонентов. Например, добавление нового поля в запись или создание нового файла никак не повлияет на работу имеющихся приложений. Однако удаление поля из используемого приложением файла повлияет на это приложение, а потому его также потребуется соответствующим образом модифицировать. И, наконец, следует объяснить последний термин из определения базы данных, а именно понятие "логически связанный". При анализе информационных потребностей потребителя следует выделить сущности, атрибуты и связи. Сущностью (entity) называется отдельный тип объекта (человек, место или вещь, понятие или событие), который нужно представить в базе данных, атрибутом (attribute) называется свойство, которое описывает некоторую характеристику описываемого объекта; связь (relationship) - это то, что объединяет несколько сущностей. Подобная база данных представляет сущности, атрибуты и логические связи между объектами. Иначе говоря, база данных содержит логически связанные данные. Данные в базе обладают некоторой структурой (например: таблицы). Но как была получена такая структура? Ответ на этот вопрос достаточно прост: структура базы данных определяется во время ее проектирования. Однако сам процесс проектирования базы данных может оказаться чрезвычайно сложным. Для создания системы, которая удовлетворяла бы информационным потребностям некоторого потребителя, необходимо использовать подход, совершенно отличающийся от методов разработки обычных, файловых систем, в которых вся работа заключается в разработке приложений, удовлетворяющим нуждам отдельных подразделений. Для успешной реализации системы на основе базы данных необходимо подумать, прежде всего, о данных и лишь потом о приложениях. Чтобы система полностью удовлетворяла запросам пользователей, необходимо очень внимательно отнестись к процессу проектирования базы данных. Плохо спроектированная база данных будет порождать ошибки, способные привести к принятию неправильных решений, которые повлекут за собой самые серьезные последствия для данной организации. С другой стороны, хорошо спроектированная база данных позволит создать систему, поставляющую корректную информацию, которая может успешно использоваться для принятия правильных и эффективных решений.

1.5.1. Логические и физические описания

Как следует из известных справочников [26,30], база данных – это именованная совокупность данных, отражающая состояние объектов и их отношений в рассматриваемой предметной области.

Если следовать Дж. Мартину [21], К. Дейту [32], Т. Тиори и Дж. Фраю [23], то организацию базы данных можно условно разделить на логическую и физическую. Логическая организация базы данных отражает представление о базе данных прикладного программиста и пользователя, а физическая – представление системного программиста и аналитика. В [32] выделяется еще один уровень, более абстрактный, чем логический, называемый концептуальным представлением и отражающий представление администратора базы данных.

Описания данных и отношений между ними бывают двух видов: логические и физические.

Физическое описание данных определяет способы физической записи данных во внешней памяти. Логическое описание данных указывает на то, в каком виде данные представляются прикладному программисту или пользователю данных. Термины логический и физический будут использоваться для описания различных аспектов данных: логический - указывает на то, как данные представляются программисту или пользователю, в то время как физический – указывает, каким образом данные записываются в среде хранения. Чтобы уменьшить объем занимаемой памяти или время доступа, физическая запись может содержать несколько логических записей. Структура данных и их взаимосвязь в физической и логической организациях данных могут не совпадать. Терминами логическое отношение, логическая структура и логическое описание данных описывается представление данных с точки зрения программиста или пользователя. Термины физическое отношение, физическая структура и физическое описание данных описывают реальные способы хранения данных. В физических записях на диске содержатся более короткие логические записи, которые связаны в цепь. Программисту требуется файл логических записей в порядке их расположения в цепи. Программное обеспечение будет передавать программе логические записи точно в требуемой последовательности. Другие программы могут получать эти записи в ином порядке. Возможны и многие другие типы различий между логической и физической организациями.

Методы физической организации данных хорошо изложены в классических книгах Д.Кнута, Дж.Мартина, А.Ахо, Дж.Хопкрофта, Дж.Ульмана.

Рассмотрим некоторые определения, используемые при логическом описании данных.

Байт – наименьшая адресуемая группа битов (обычно 8 битов).

Элемент данных – наименьшая единица поименованных данных (любое количество битов или байтов), часто элемент данных называют полем.

Агрегат данных – поименованная совокупность элементов данных внутри записи, рассматриваемая как единое целое (агрегат данных ДАТА может состоять из элементов данных МЕСЯЦ, ДЕНЬ, ГОД).

Запись - поименованная совокупность элементов данных или агрегатов данных. При чтении из базы данных прикладная программа может полностью прочитать логическую запись.

Файл – поименованная совокупность всех экземпляров логических записей заданного типа.

База данных – совокупность экземпляров различных типов записей и отношений между записями, агрегатами данных, элементами данных.

Форма физического хранения данных существенно отличается от их логического представления. Рассмотрим некоторые определения, используемые при описании физического хранения данных.

Физическая запись – элементарная единица данных, которая может быть считана или записана одной командой ввода-вывода компьютера.

Блок записей – группа данных, составляющих физическую запись.

Набор данных – поименованная совокупность физических записей, также включает данные, используемые для определения местоположения записей (например, индексы), может содержаться внутри одного тома или размещаться на многих томах.

1.5.2. Требования к организации базы данных

Опираясь на вышеприведенные определения сути понятия базы данных можно выделить следующие требования к организации базы данных и ее обработке:

1. отсутствие дублирования данных в различных объектах модели данных, обеспечивающее однократный ввод данных и простоту внесения изменений данных;
2. непротиворечивость и целостность данных;
3. возможность многоаспектного доступа, всевозможные выборки из массивов недублированной информации и их использование различными задачами и приложениями пользователя;
4. защита и восстановление данных при аварийных ситуациях, аппаратных и программных сбоях, ошибках пользователя;
5. защита данных от несанкционированного доступа с использованием паролей и другими средствами разграничения доступа для различных пользователей;
6. возможность модификации структуры БД без повторной загрузки данных;
7. обеспечение независимости программ от данных, позволяющей сохранить программы при модификации структуры БД, физической реорганизации данных.

Таким образом, базу данных (БД) можно определить как совокупность интегрированных, не дублированных и логически взаимосвязанных данных, организованных на машинном носителе средствами СУБД в соответствии со структурами данных и моделью, которые она поддерживает.

Дублирование данных означает повторение одних и тех же не ключевых данных в разных массивах (таблицах). Целостность и непротиворечивость данных означает такое наполнение базы данными, при котором все записи из разных массивов имеют корректные логические связи с записями других массивов, в случае если такие связи определены в логической структуре БД. Однократность ввода данных означает, что при вводе идентификационных данных их не приходится вводить в логически связанные массивы повторно. Модель данных определяет способ логической организации данных, реализуемой СУБД. Модели данных, поддерживаемые в различных СУБД, - иерархические, сетевые, реляционные, обеспечивают отображение концептуальной модели предметной области в структуры данных выбранной СУБД. Независимость программ от данных означает возможность модификации структуры БД, без необходимости переработки программ обработки баз данных. Важнейшим условием корректного построения структуры БД является разработка концептуальной модели, отражающей логическую структуру объектов предметной области.

Дж. Мартин [21] определяет базу данных, как совокупность взаимосвязанных хранящихся вместе данных при наличии такой минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений;

данные запоминаются так, чтобы они были независимы от программ, использующих эти данные; для добавления новых или модификации существующих данных, а также для поиска данных в базе данных применяется общий управляемый способ. Говорят, что система содержит совокупность баз данных, если эти базы данных структурно полностью самостоятельны.

В заключении приведем некоторые определения из ГОСТ 34.321-96 «Эталонная модель управления данными».

База данных (database) – совокупность взаимосвязанных данных, организованных в соответствии со схемой базы данных таким образом, чтобы с ними мог работать пользователь.

Схема базы данных – формальное описание данных в соответствии с конкретной схемой данных.

Схема данных – логическое представление организации данных.

1.5.3. Структурирование данных

Создавая базу данных, пользователь стремится упорядочить информацию по различным признакам и быстро извлекать выборку с произвольным сочетанием признаков. Сделать это возможно, только если данные структурированы. Структурирование - это введение соглашений о способах представления данных.

Неструктурированными называют данные, записанные, например, в текстовом файле (см. рис. 1.3).

Личное дело N 16493, Сергеев Петр Михайлович, дата рождения 1 января1976г.;Л/дN16593,Петрова Анна Владимировна, дата рожд. 15 марта 1975 г.;Nличн.дела16693,д.р.14.04.76,Анохин Андрей Борисович

Рис. 1.3. Неструктурированные данные

Чтобы автоматизировать поиск и систематизировать эти данные, необходимо выработать определенные соглашения о способах представления данных, т.е. дату рождения нужно записывать одинаково для каждого студента, она должна иметь одинаковую длину и определенное место среди остальной информации (см. рис. 1.4).

Пользователями базы данных могут быть различные прикладные программы, а также специалисты предметной области, выступающие в роли потребителей. В современной технологии баз данных предполагается, что создание базы данных, ее поддержка и обеспечение доступа пользователей к ней осуществляются централизованно с помощью специального программного инструментария - системы управления базами данных.

№личного дела	Фамилия	Имя	Отчество	Дата рождения
16493	Сергеев	Петр	Михайлович	01.01.76
16593	Петрова	Анна	Владимировна	15.03.75
16693	Анохин	Андрей	Борисович	14.04.76

Рис.1.4. Структурированные данные

База данных (БД) - это поименованная совокупность структурированных данных, относящихся к определенной предметной области. Система управления базами данных (СУБД) - это комплекс программных и языковых средств, необходимых для создания баз данных, поддержания их в актуальном состоянии и организации поиска в них необходимой информации. Централизованный характер управления данными в базе данных предполагает необходимость существования некоторого лица (группы лиц), на которое возлагаются функции администрирования данными, хранимыми в базе.

• Структурные элементы базы данных. Понятие базы данных тесно связано с такими понятиями структурных элементов, как поле, запись, файл (таблица) (рис.1.5)..

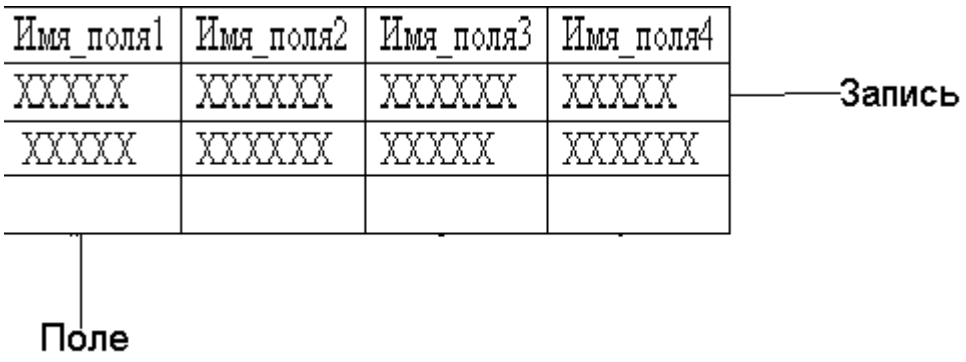

Рис. 1.5. Структурные элементы базы данных

Поле - элементарная единица логической организации данных, которая соответствует неделимой единице информации - реквизиту. Для описания поля используют следующие характеристики:

• имя, например, Фамилия, Имя, Отчество, Дата рождения;

• тип, например, символьный, числовой, календарный;

• длина, например, 15 байт, причем будет определяться максимально возможным количеством символов;

• точность для числовых данных, например два десятичных знака для отображения дробной части числа

Запись - совокупность логически связанных полей. Экземпляр записи - отдельная реализация записи, содержащая конкретные значения ее полей.

Файл (таблица) - совокупность экземпляров записей одной структуры. Описание логической структуры записи файла содержит последовательность расположения полей записи и их основные характеристики, как это показано на рис.1.6.

Имя файла					
Поле		Признак ключа	Формат поля		
Имя (обозначение)	Полное наименование		Тип	Длина	Точность (для чисел)
имя1					
имя n					

Рис.1.6. Описание логической структуры записи файла

В структуре записи файла указываются поля, значения которых являются ключами: первичными (ПК), которые идентифицируют экземпляр записи, и вторичными (ВК), которые выполняют роль поисковых или группировочных признаков (по значению вторичного ключа можно найти несколько записей). На рис.1.7 приведен пример описания логической структуры записи файла (таблицы) СТУДЕНТ, содержимое которого приводится на рис.1.4.

Имя файла СТУДЕНТ					
Поле		Признак ключа	Формат поля		
Обозначение	Наименование		Тип	Длина	Точность
№	Номер личного дела	/	Симв.	5	
Фамилия	Фамилия студента		Симв.	15	
Имя	Имя студента		Симв.	10	
Отчество	Отчество студента		Симв.	15	
Дата	Дата рождения		Симв.	8	

Рис.1.7. Пример описания логической структуры записи файла (таблицы) СТУДЕНТ

Структура записи файла СТУДЕНТ линейная, она содержит записи фиксированной длины. Повторяющиеся группы значений полей в записи отсутствуют. Обращение к значению поля производится по его номеру.

1.6. Классификация баз данных

По технологии обработки данных базы данных подразделяются на централизованные и распределенные.

Централизованная база данных хранится в памяти одной вычислительной системы. Эта вычислительная система может быть мэйнфреймом - тогда доступ к ней организуется с использованием терминалов - или файловым сервером локальной сети ПК.

Распределенная база данных состоит из нескольких, возможно, пересекающихся или даже дублирующих друг друга частей, которые хранятся в различных ЭВМ вычислительной сети. Работа с такой базой осуществляется с помощью системы управления распределенной базой данных (СУРБД).

Основная задача систем управления распределенными базами данных состоит в обеспечении средства интеграции локальных баз данных, располагающихся в некоторых узлах вычислительной сети, с тем, чтобы пользователь, работающий в любом узле сети, имел доступ ко всем этим базам данных как к единой базе.

Возможны однородные и неоднородные распределенные базы данных. В однородном случае каждая локальная база данных управляется одной и той же СУБД. В неоднородной системе локальные базы данных могут относиться даже к разным моделям данных. Сетевая интеграция неоднородных баз данных - очень сложная проблема. Многие решения известны на теоретическом уровне, но пока не удается справиться с главной проблемой: недостаточной эффективностью интегрированных систем. Более успешно решается

промежуточная задача - интеграция неоднородных SQL-ориентированных систем. Этому в большой степени способствует стандартизация языка SQL.

По способу доступа к данным базы данных разделяются на базы данных с локальным доступом и базы данных с сетевым доступом.

Для всех современных баз данных можно организовать сетевой доступ с многопользовательским режимом работы.

Рис. 1.8. Схема работы с БД в локальной сети с выделенным файловым сервером

Централизованные базы данных с сетевым доступом могут иметь следующую архитектуру:

- файл-сервер;
- клиент-сервер базы данных;
- "тонкий клиент" - сервер приложений - сервер базы данных (трехуровневая архитектура).

Файл-сервер. Архитектура систем БД с сетевым доступом предполагает выделение одной из машин сети в качестве центральной (файловый сервер). На этот компьютер устанавливается операционная система (ОС) для выделенного сервера (например, Microsoft Windows Server 2003). На нем же хранится совместно используемая централизованная БД в виде одного или группы файлов. Все другие компьютеры сети выполняют функции рабочих станций (могут работать в ОС Microsoft Windows 2000 Professional или Microsoft Windows 98). Файлы базы данных в соответствии с пользовательскими запросами передаются на рабочие станции, где и производится обработка информации (рис. 1.8). При большой интенсивности доступа к одним и тем же данным производительность информационной системы падает. Пользователи могут создавать также локальные БД на рабочих станциях.

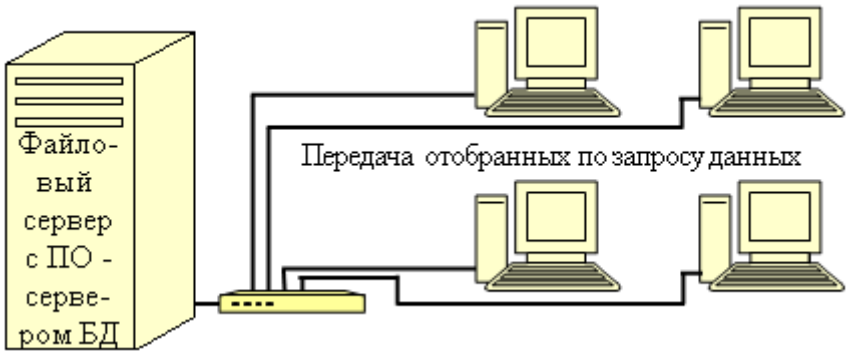

Рис. 1.9. Схема работы с БД в архитектуре «Клиент-сервер»

Клиент-сервер. В этой архитектуре на выделенном сервере, работающем под управлением серверной операционной системы, устанавливается специальное программное обеспечение (ПО) - сервер БД, например, Microsoft SQL Server или Oracle. СУБД подразделяется на две части: клиентскую и серверную. Основа работы сервера БД - использование языка запросов (SQL). Запрос на языке SQL, передаваемый клиентом (рабочей станцией) серверу БД, порождает поиск и извлечение данных на сервере. Извлеченные данные транспортируются по сети от сервера к клиенту (рис. 1.9). Тем самым, количество передаваемой по сети информации уменьшается во много раз.

Трехуровневая архитектура функционирует в Интранет- и Интернет-сетях. Клиентская часть ("тонкий клиент"), взаимодействующая с пользователем, представляет собой HTML-страницу в Web-браузере либо Windows-приложение, взаимодействующее с Web-сервисами. Вся программная логика вынесена на сервер приложений, который обеспечивает формирование запросов к базе данных, передаваемых на выполнение серверу баз данных. Сервер приложений может быть Web-сервером или специализированной программой (например, Oracle Forms Server) (рис. 1.10).

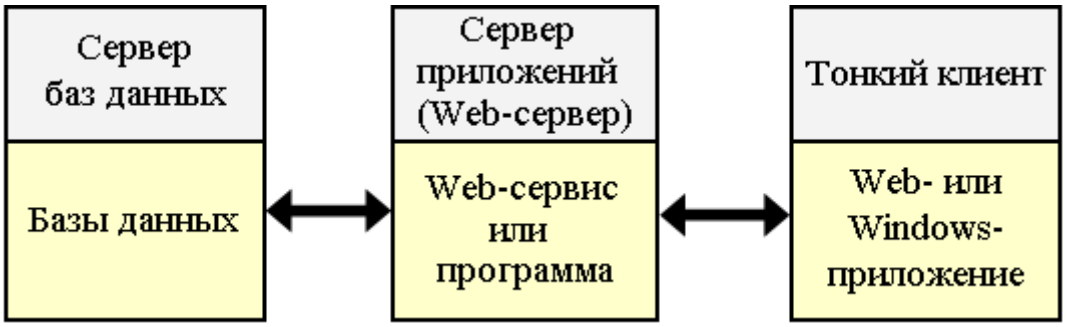

Рис. 1.10. Схема работы с БД в трехуровневой архитектуре

По типу модели данных, поддерживаемой конкретной СУБД, базы данных делятся на иерархические, сетевые, реляционные.

1.7. Модели данных

Модель данных – интегрированный набор понятий для описания данных, связей между ними и ограничений, накладываемых на данные в некоторой предметной области. (М.Р.Когаловский, «Энциклопедия технологий баз данных»)

Проблематика моделирования данных связана с таким представлением данных, которое наиболее естественно отражает реальный мир и может поддерживаться компьютерными средствами. При этом эти два требования часто противоречат друг другу. Для того, чтобы определить наилучшую организацию данных (с точки зрения конкретного приложения), необходимо установить, какие характеристики данных важны для выражения сущности их значения. Это позволяет сформировать непротиворечивый набор общих формальных утверждений, характеризующих организацию и обработку данных, который и определяет модель данных.

Таким образом, модель данных – это интеллектуальное средство, позволяющее реализовывать интерпретацию данных в соответствии с указанными требованиями. Цель построения модели данных заключается в представлении данных в понятном виде, которое будет применимо при проектировании базы данных.

Для отображения архитектуры ANSI-SPARC можно идентифицировать следующие три связанные модели:

- внешнюю модель данных, отображающую представления каждого существующего в организации типа пользователей (описание предметной области);
- концептуальную модель данных, отображающую логическое (или обобщенное) представление о данных, не зависимое от типа выбранной СУБД;
- внутреннюю модель данных, отображающую концептуальную схему определенным образом, понятным выбранной целевой СУБД.

На сегодня в литературе предлагается множество моделей данных, они подразделяются на три категории: объектные (object-based), модели данных на основе записей (record-based) и физические модели данных. Объектные и модели данных на основе записей используются для описания данных на концептуальном и внешнем уровнях, а физические – на внутреннем уровне.

При построении объектных моделей данных используются понятия сущности, атрибутов и связей. Наиболее общие типы этих моделей: модель ”сущность-связь” или ER-модель, семантическая, функциональная, объектно-ориентированная. В настоящее время ER-модель стала одним из основных методов концептуального проектирования баз данных. Объектно-ориентированная модель расширяет определение сущности с целью включения в него не только атрибутов, описывающих состояние объекта, но и действий, которые с ним связаны, т.е. его поведение. В таком случае говорят, что объект инкапсулирует состояние и поведение.

В модели на основе записей база данных состоит из нескольких записей фиксированного формата, которые могут иметь разные типы. Каждый тип записи определяет фиксированное число полей, каждое поле имеет фиксированную длину. Выделяют три основных типа логических моделей данных на основе записей: реляционная модель данных, сетевая и иерархическая. Так как сетевая и иерархическая модели данных были созданы почти на десять лет раньше реляционной их связь с концепциями традиционной обработки файлов более очевидна. Соответственно говорят об иерархических, сетевых и реляционных СУБД, поддерживающих соответствующие модели данных.

Сетевая модель. Если в модели каждый порожденный элемент может иметь более одного исходного, то такая модель называется сетевой. В ней элемент может быть связан с любым другим, без каких либо ограничений. Сетевая БД состоит из набора записей, соответствующих каждому экземпляру объекта предметной области, и набора связей между ними. Например, информация об участии сотрудников в проектах организации может быть представлена в виде сетевой БД (рис.1.11). В данном примере сетевая модель отражает то, что в проекте могут участвовать разные сотрудники и в то же время сотрудник может участвовать в разных проектах.

Сетевые модели также создавались для мало ресурсных ЭВМ. Это достаточно сложные структуры, состоящие из "наборов" – поименованных двухуровневых деревьев. "Наборы" соединяются с помощью "записей-связок", образуя цепочки и т.д. При разработке сетевых моделей было выдумано множество "маленьких хитростей", позволяющих увеличить производительность СУБД, но существенно усложнивших последние.

Прикладной программист должен знать массу терминов, изучить несколько внутренних языков СУБД, детально представлять логическую структуру базы данных для осуществления навигации среди различных экземпляров, наборов, записей и т.п. Один из разработчиков операционной системы UNIX сказал "Сетевая база – это самый верный способ потерять данные".

Иерархическая модель позволяет строить базы данных с иерархической древовидной структурой. Эта структура определяется как дерево, образованное попарными связями.

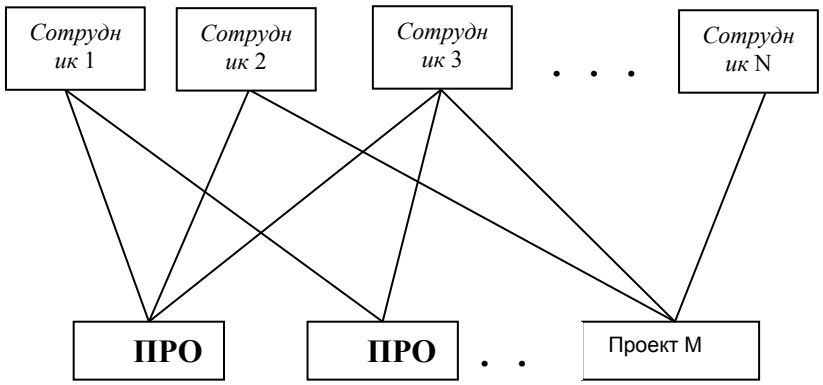

Рис.1.11. Пример сетевой структуры

На самом верхнем уровне дерева имеется один узел, называемый корнем. Все другие узлы, кроме корня, связываются только с одним узлом на более высоком по отношению к ним самим уровне. Например, на рис.1.12 объект “Организация” - предок для объектов “Отделы” и “Филиалы”. Основное достоинство таких моделей – простота описания иерархических структур реального мира.

Простота организации иерархической модели, наличие заранее заданных связей между записями, сходство с физическими моделями данных позволяли добиваться приемлемой производительности иерархических СУБД на медленных ЭВМ с весьма ограниченными объемами памяти. Но, если данные не имели древовидной структуры, то возникала масса сложностей при построении иерархической модели и желании добиться нужной производительности.

Сложность практического использования иерархических и сетевых СУБД заставляла искать иные способы представления данных. В конце 60-х годов появились СУБД на основе инвертированных файлов, отличающиеся простотой организации и наличием весьма удобных языков манипулирования данными. Однако такие СУБД обладают рядом ограничений на количество файлов для хранения данных, количество связей между ними, длину записи и количество ее полей.

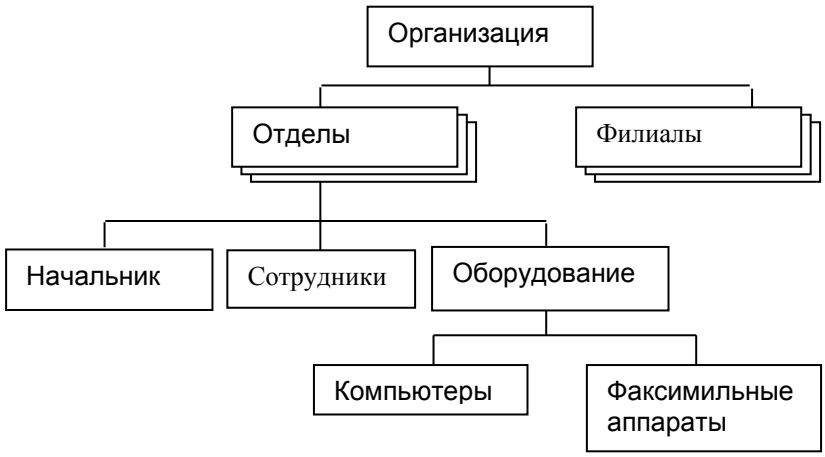

Рис.1.12. Пример иерархической древовидной структуры

Концепция реляционной модели была предложена Коддом в 1970 году. В основе реляционной модели данных лежит понятие отношение (relation). Отношение удобно представляется в виде двумерной таблицы при соблюдении определенных ограничивающих условий. Таблица понятна, обозрима и привычна для использования. Строки таблицы представляют собой отдельные записи – перечень элементов сущности. Столбцы – свойства сущности или атрибуты. При этом одни таблицы служат для описания объектов (объектные отношения), другие – для описания связей между ними (связные отношения).

Оригинальность подхода Кодда состояла в том, что он предложил применять к отношениям стройную систему операций, позволяющих получать одни отношения из других и тем самым разделить информацию на хранимую и вычисляемую. Такой подход не только упрощал представление информации, но и позволял экономить память: в случае необходимости часть информации можно было получать путем вычислений.

Например, для хранения сущности «студент» используется объектное отношение СТУДЕНТ, в котором свойства сущности располагаются в столбцах таблицы:

СТУДЕНТ

Ф. И. О.	Дата рождения	Курс	Специальность
Иванов И.И.	12.03.80	4	История
Петров П.П.	23.08.81	3	Физика
Сидоров С.С.	02.10.82	2	Педагогика

Столбцы отношения называют атрибутами и присваивают им имена. Значения конкретного атрибута выбираются из домена – множества всех возможных значений атрибутов объекта.

В объектном отношении один или несколько атрибутов однозначно идентифицируют отдельный объект. Такой атрибут или совокупность атрибутов называют ключом отношения. В объектном отношении не должно быть строк с одинаковыми ключами. Это одно из требований целостности данных.

Связное отношение хранит ключи двух или более отношений, то есть по ключам устанавливаются связи между объектами отношений. Рассмотрим связное отношение ИЗУЧАЕТ (студент, предмет). При этом в базе данных имеются объектные отношения СТУДЕНТ (Ф.И.О., Курс, Специальность) и ПРЕДМЕТ (Название, Число семестров) со следующими данными:

СТУДЕНТ

Ф. И. О.	Курс	Специальность
Иванов И.И.	4	История
Петров П.П.	3	Физика
Сидоров С.С.	2	Педагогика

ПРЕДМЕТ

Название	Число семестров
История	4
Иностранный язык	3
Вычислительная техника	2
Политология	2

Связное отношение ИЗУЧАЕТ может содержать такие данные:

ИЗУЧАЕТ

Студент	Предмет
Иванов И.И.	История
Иванов И.И.	Политология
Петров П.П.	Вычислительная техника
Сидоров С.С.	Иностранный язык

Связное отношение кроме связываемых ключей может иметь и другие атрибуты:

ИЗУЧАЕТ

Студент	Предмет	Оценка
Иванов И.И.	История	5
Иванов И.И.	Политология	4
Петров П.П.	Вычислительная техника	4
Сидоров С.С.	Иностранный язык	5

Ключи в связных отношениях называются внешними ключами, поскольку они являются первичными в других отношениях. С помощью внешних ключей поддерживается взаимосвязь таблиц в реляционной модели.

В настоящее время на рынке информационных услуг доминируют реляционные модели, что обусловлено рядом причин:

1. наличие развитой теории реляционной модели данных, которая поддерживается теоретическими исследованиями в большей степени по сравнению с другими моделями;
2. наличие аппарата сведения к реляционной других моделей данных;
3. поддержка реляционной моделью специальных средств ускоренного доступа к информации;
4. возможность манипулирования данными без необходимости знания конкретной физической организации БД во внешней памяти;
5. наличие стандартизованного высокоуровневого языка запросов к базам данных.

Реляционная модель данных основана на понятии математических отношений, в ней данные и связи представлены в виде таблиц, каждая из которых имеет несколько столбцов с уникальными именами. Однако такое представление относится только к логической структуре базы данных (внешний и концептуальный уровни), но не относится к физической структуре базы данных, которая может быть реализована с помощью разнообразных структур хранения.

Семантическая модель (Диаграмма "сущность-связь"). (Entity-Relationship diagrams, или ER- diagram) служит для описания схемы базы на концептуальном уровне проектирования. Метод был предложен в 1976 г. Питером Пин Шань Ченом (Peter Pin Shan Chen) [49,50]. На диаграммах "сущность-связь" сущности изображаются в виде прямоугольников, атрибуты - в виде эллипсов, а связи - в виде ромбов (рис. 1.13).

В дальнейшем многими авторами были разработаны свои варианты подобных моделей (нотация Мартина, нотация IDEF1X, нотация Баркера и др.). Кроме того, различные программные средства, реализующие одну и ту же нотацию, могут отличаться своими возможностями. По сути, все варианты диаграмм "сущность-связь" исходят из одной идеи - рисунок всегда нагляднее текстового описания. Все такие диаграммы используют

графическое изображение сущностей предметной области, их свойств (атрибутов), и взаимосвязей между сущностями.

Элементами описания модели данных на концептуальном уровне являются сущности, атрибуты, домены и связи.

Сущность - некоторый обособленный объект или событие, информацию о котором необходимо сохранять в базе данных, имеющий определенный набор свойств - атрибутов. Сущности могут быть как физические (реально существующие объекты: например, ПРОДАВЕЦ, атрибуты - № продавца, фамилия, имя, отчество и т.д.), так и абстрактные (например, ПРОДАЖА, атрибуты - товар, дата, продавец и пр.). Для сущностей различают ее тип и экземпляр. Тип характеризуется именем и списком свойств, а экземпляр - конкретными значениями свойств.

Атрибуты сущности бывают: идентифицирующие и описательные. Идентифицирующие атрибуты имеют уникальное значение для сущностей данного типа и являются потенциальными ключами. Они позволяют однозначно распознавать экземпляры сущности. Из потенциальных ключей выбирается один первичный ключ (ПК). В качестве ПК обычно выбирается потенциальный ключ, по которому чаще происходит обращение к экземплярам записи. ПК должен включать в свой состав минимально необходимое для идентификации количество атрибутов. Остальные атрибуты называются описательными.

Простые и составные: простой атрибут состоит из одного компонента, его значение неделимо, составной атрибут является комбинацией нескольких компонентов, возможно, принадлежащих разным типам данных (например, адрес). Решение о том, использовать составной атрибут или разбивать его на компоненты, зависит от особенностей процессов его использования и может быть связано с обеспечением высокой скорости работы с большими базами данных.

Однозначные и многозначные: могут иметь соответственно одно или много значений для каждого экземпляра сущности.

Основные и производные: значение основного атрибута не зависит от других атрибутов, значение производного атрибута вычисляется на основе значений других атрибутов (например, возраст человека вычисляется на основе даты его рождения и текущей даты

Спецификация атрибута состоит из его названия, указания типа данных и описания ограничений целостности - множества значений (или домена), которые может принимать данный атрибут.

Домен - это набор всех допустимых значений, которые может содержать атрибут. Понятие "домен" часто путают с понятием "тип данных". Необходимо различать эти два понятия. Тип данных - это физическая концепция, а домен - логическая. Например, "целое число" - это тип данных, а "возраст" - это домен.

Связи - на концептуальном уровне представляют собой простые ассоциации между сущностями. Например, утверждение "Покупатели приобретают продукты" указывает, что между сущностями "Покупатели" и "Продукты" существует связь, и такие сущности называются участниками этой связи. Существует несколько типов связей между двумя сущностями: "один к одному", "один ко многим" и "многие ко многим".

Каждая связь в ER - модели характеризуется именем, обязательностью, типом и степенью. Различают факультативные и обязательные связи. Если сущность одного типа оказывается по необходимости связанной с сущностью другого типа, то между этими типами объектов существует обязательная связь. Иначе связь является факультативной.

Степень связи определяется количеством сущностей, которые охвачены данной связью. Пример бинарной связи - связь между отделом и сотрудниками, которые в нем работают.

Физические модели данных описывают то, как данные хранятся в компьютере, представляя информацию о структуре записей, их упорядоченности и существующих путях

доступа. При физической организации баз данных мы имеем дело не с представлением данных в прикладных программах, а с их размещением на запоминающих устройствах.

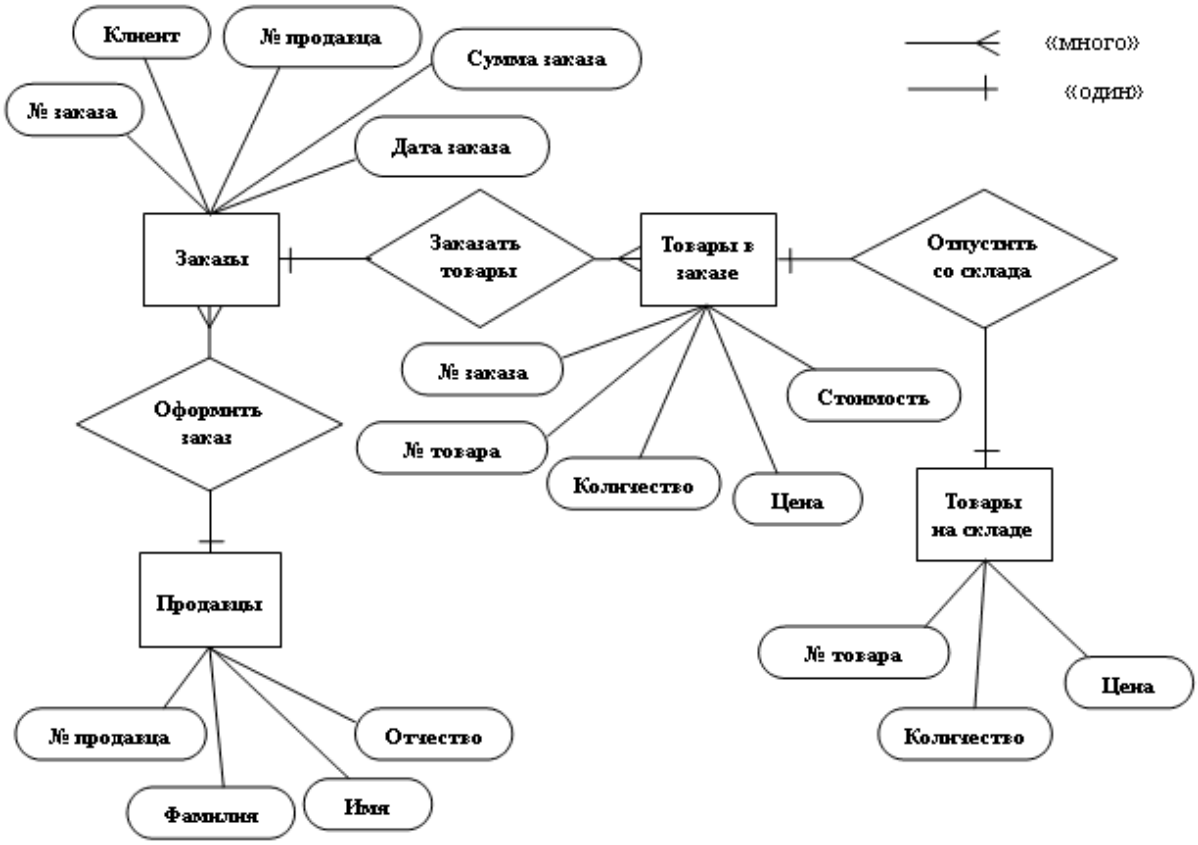

Рис. 1.13. Диаграмма "сущность-связь"

Критерии, определяющие выбор физической организации, отличаются от тех, которые определяют выбор логической организации данных. При выборе физической организации решающим фактором является эффективность, причем согласно Дж. Мартину [21] на первом месте стоит обеспечение эффективности поиска, далее идут эффективность операций занесения и удаления и затем обеспечение компактности данных. Кроме того, в последнее время большую актуальность приобрели проблемы защиты данных от несанкционированного доступа.

1.8. Об ограничениях целостности

Целостность (от англ. integrity - нетронутость, неприкосновенность, сохранность, целостность) понимается как правильность данных в любой момент времени. Поддержание целостности базы данных может рассматриваться как защита данных от неверных изменений или разрушений.

Выделяют три группы правил целостности:

1. Целостность по сущностям.
2. Целостность по ссылкам.
3. Целостность, определяемая пользователем.

Общая мотивировка первых двух правил целостности общих для любых реляционных баз данных, состоит в следующем:

• Не допускается, чтобы какой-либо атрибут, участвующий в первичном ключе, принимал неопределенное значение.

• Для каждого внешнего ключа в проекте проектировщик базы данных должен специфицировать не только атрибут или комбинацию атрибутов, составляющих этот внешний ключ, и целевое отношение, которое идентифицируется этим ключом, но также и ответы на три вопроса (три ограничения, которые относятся к этому внешнему ключу).

• Для любой конкретной базы данных существует ряд дополнительных специфических правил, которые относятся к ней одной и определяются разработчиком. Чаще всего контролируется:

– уникальность тех или иных атрибутов;

– диапазон значений (экзаменационная оценка от 2 до 5);

– принадлежность набору значений (пол "М" или "Ж").

Может ли внешний ключ принимать неопределенное значение (NULL-значение)? Значение внешнего ключа должно:

• либо быть равным значению первичного ключа отношения, с которым он связан;

• либо быть полностью неопределенным, при этом каждое значение атрибута, участвующего во внешнем ключе, должно быть неопределенным.

Например, в отношении Поставка, очевидно, поставка, осуществляемая неизвестным поставщиком, или поставка неизвестного продукта, не может иметь NULL-значения, в то время как атрибут Отдел в отношении Сотрудник может иметь NULL-значение, если сотрудник пока еще не зачислен ни в какой отдел.

Что должно случиться при попытке УДАЛЕНИЯ экземпляра целевой сущности, на которую ссылается внешний ключ? Например, при удалении поставщика, который осуществил по крайней мере одну поставку.

Существует три возможности:

КАСКАДИРУЕТСЯ - Операция удаления "каскадируется" с тем, чтобы удалить также поставки этого поставщика.

ОГРАНИЧИВАЕТСЯ - Удаляются лишь те поставщики, которые еще не осуществляли поставок. Иначе операция удаления отвергается.

УСТАНАВЛИВАЕТСЯ - Для всех поставок удаляемого поставщика внешний ключ устанавливается в неопределенное значение, а затем этот поставщик удаляется. Такая возможность, конечно, неприменима, если данный внешний ключ не должен содержать NULL-значений.

Что должно происходить при попытке ОБНОВЛЕНИЯ первичного ключа экземпляра целевой сущности, на которую ссылается некоторый внешний ключ? Например, может быть предпринята попытка обновить номер такого поставщика, для которого имеется, по крайней мере, одна соответствующая поставка. Имеются те же три возможности, как и при удалении:

КАСКАДИРУЕТСЯ - Операция обновления "каскадируется" с тем, чтобы обновить также и внешний ключ в поставках этого поставщика.

ОГРАНИЧИВАЕТСЯ - Обновляются первичные ключи лишь тех поставщиков, которые еще не осуществляли поставок. Иначе операция обновления отвергается.

УСТАНАВЛИВАЕТСЯ - Для всех поставок такого поставщика внешний ключ устанавливается в неопределенное значение, а затем обновляется первичный ключ поставщика. Такая возможность, конечно, неприменима, если данный внешний ключ не должен содержать NULL-значений.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

1. Предпосылки разработки стандартов технологий баз данных и отдельных их элементов.
2. Дайте характеристику периода становление новых технологий, связанных с созданием, поддержкой и использованием баз данных.
3. Как проходило формирование основ методологии построения систем баз данных.
4. Дайте характеристику формирования теоретических основ современных технологий баз данных.
5. Стандарты баз данных, их назначение и виды. Стандарты открытых систем. Организации по стандартизации технологий баз данных.
6. Стандарт языка SQL
7. ODMG и стандарты объектных баз данных
8. Стандарт ODMG-93
9. Стандарт ODMG 2.0
10. Стандарт ODMG 3.0
11. Стандарты API для систем баз данных
12. Охарактеризуйте понятия: база данных, схема базы данных, метаданные, словарь данных.
13. Что такое жизненный цикл баз данных? Этапы ЖЦ.
14. Дайте характеристику архитектуры ANSI/SPARC.
15. Опишите три уровня абстракции данных.
16. Какие требования предъявляются к организации баз данных?
17. Что такое структурирование?
18. Каким образом можно классифицировать базы данных?
19. Что такое модель? Характеристика иерархической модели данных.
20. Характеристика сетевой модели данных.
21. Характеристика реляционной модели данных.
22. Характеристика модели «сущность-связь».
23. Как осуществляется поддержание целостности базы данных?

ГЛАВА 2. СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

2.1. Основы систем управления базами данных

На заре развития вычислительной техники, когда мощности вычислительных машин были несравнимы с нынешними, вычислительные машины использовались в первую очередь для числовой обработки данных. Большинство вычислительных задач, как правило, легко уложить в следующую схему:

- задача читает исходные данные
- задача преобразует считанные данные в соответствии с алгоритмом обработки
- задача записывает выходные данные (например, печатает их на принтер)

При такой структуре задач способ хранения данных не имеет для вычислительных машин решающего значения – важно лишь, чтобы все данные были введены в машину в нужном количестве и в определенном порядке. Данные для таких машин обычно хранились в виде колоды перфокарт или перфоленты: одна колода или одна перфолента на одну группу данных. Способ организации данных – последовательный – проистекал из самого физического способа их хранения. Появившиеся позднее накопители на магнитной ленте просто «узаконили» этот способ хранения в виде последовательного файла или же набора подряд идущих последовательных файлов (емкость магнитной ленты позволяла хранить на ней более одного файла). Как правило, иных способов хранения данных для вычислительных задач в большинстве случаев не требуется – не случайно в язык ФОРТРАН, являющийся традиционным средством описания вычислительных задач, работа с файлами прямого доступа была введена лишь в 70-х годах.

Файлы прямого доступа появились в вычислительной практике с появлением накопителей на магнитных дисках и магнитных барабанах. Конструкция этих накопителей такова, что позволяет сравнительно быстро получить доступ к любому участку файла (здесь, как в накопителях на магнитной ленте, не требуется длительная перемотка к нужной части). Эта возможность оказалась востребованной в задачах обработки коммерческой информации, в частности, в языке КОБОЛ. Характерное свойство коммерческих задач – невысокая сложность алгоритмов обработки и чрезвычайно большой объем разнородной информации, подлежащей обработке. Необходимость различного рода выборок по имеющейся информации не позволяет в таких задачах пользоваться последовательным просмотром данных без существенного снижения производительности системы. Именно коммерческие задачи породили потребность в новом способе организации данных – файлам прямого доступа.

Следующий этап развития – появление файлов с индексно-последовательной организацией данных, – также стимулировался потребностями коммерческой обработки информации. В файлах данных этого типа впервые появилось понятие «ключа» - особого знака, приписанного каждой записи в файле, который позволяет найти эту запись не по ее местоположению в файле, а по содержимому ключа. Индексно-последовательные файлы допускают как последовательный просмотр содержимого, так и произвольный порядок выборки данных из файла в соответствии с заданным ключом. С появлением понятия «ключ» файлы данных обрели свойство ассоциативности (возможность доступа к данным по содержимому, а не по адресу).

Индексно-последовательные файлы можно считать предтечами современных баз данных. С момента их появления начались интенсивные работы в области организации и хранения данных, и тогда же проблемы хранения и быстрого доступа к данным начали перетекать с уровня компонент операционной системы (файловая подсистема) на уровень специализированного программного обеспечения, не являющегося частью операционной системы, а выполняющего самостоятельные функции по организации и хранению

информации. Здесь важной вехой следует считать появление т.н. модели данных, т.е. фактически способа представления данных внутри файла данных. Из всех моделей наибольшую известность получили следующие: иерархическая (древовидная) модель, сетевая модель, реляционная модель

Иерархическая модель является в какой-то мере эволюцией индексно-последовательных файлов. В самом деле, подавляющее большинство коммерческих документов (счета, накладные, журналы проводок) имеют иерархическую структуру. Они, как правило, состоят из шапки документа, содержащей набор реквизитов (или полей), причем одно или несколько полей являются ключевыми (например, номер и дата выписки документа), т.е. однозначно идентифицируют этот документ среди подобных. Внутри документа, кроме шапки, имеется одна или несколько строк документа, также состоящих из набора реквизитов (полей), один или несколько из которых являются ключевыми (например, порядковый номер строки), т.е. однозначно идентифицируют строку в пределах данного документа. В иерархических моделях данных впервые появилось и понятие «метаданные», т.е. описание способа представления данных внутри машины. Наличие метаданных делает возможным исследование структуры данных (хотя, разумеется, оно предназначалось не для этого), а также дает качественно новую возможность – получать доступ к данным по их именам, а не по их физическому местоположению в файле. Это означает, в частности, что при дополнении данных новыми реквизитами (полями) отпадает необходимость переписывать все ранее написанное программное обеспечение под изменившуюся структуру данных

Во многих коммерческих документах есть ссылка на другие документы (например, в счете-фактуре может быть ссылка на номера товаросопроводительных документов). Ссылки на другие документы нарушают строгую иерархическую модель. Расширение и обобщение понятия «ссылка» привело к новому способу представления данных – сетевой модели. При сетевом способе представления коммерческий документ разбивается на отдельные части, каждая из которых представляет собой запись (например, запись «шапка документа», запись «строка документа»), связанная ссылками с другими записями. Наличие ссылок позволяет организовывать сколь угодно сложную структуру данных, но работа с ними далеко не так «прозрачна», как работа с иерархически организованными данными. Видимо, это послужило одной из причин, по которым сетевые модели данных не получили широкого распространения.

В начале 70-х годов У.Коддом была разработана новая модель представления данных – реляционная. Ссылки в этой модели присутствуют в неявном виде, как одна из операций (соединение таблиц). В силу ряда причин как объективного, так и исторического характера, именно эта модель данных является в настоящее время стандартом де-факто описания данных. Одним из ее достоинств, в частности, является возможность строгого математического описания, тогда как две предыдущие модели базировались более на интуитивной понятности, чем на строгом математическом аппарате.

Модель представления данных – не единственное, что отличает современные средства хранения и доступа к данным. В начале 70-х годов изменился сам стиль работы с вычислительными машинами. Если раньше они представляли собой «ящик» по переработке информации, к которому с одной стороны подносят колоды перфокарт, а с другой – вытаскивают километры распечаток, то с появлением алфавитно-цифровых дисплеев за машину смогли сесть не только «специально обученные» операторы, но и обычные пользователи – кладовщики, бухгалтеры, начальники различных рангов и т.д. Возникло понятие «режим разделения времени». Одновременно с этим возникли и проблемы в работе с данными, а именно проблемы одновременного доступа к данным и проблемы целостности данных (под целостностью данных понимается соответствие хранящихся данных определенным критериям – например, в системе не могут храниться строки документа, если к ним нет шапки документа). Если ранее такие проблемы не возникали в силу

последовательной обработки информации, то теперь несколько пользователей могли одновременно изменять содержимое одного и того же документа (например, отпускать со склада одну и ту же продукцию) и программное обеспечение должно было обеспечивать предсказуемость результатов таких изменений. Первым способом решения этих проблем стало введение механизма блокировок - по сути, придание изменениям данных строго последовательного характера, т.е. проблемы, к примеру, отпуска со склада продукции в количестве, большем, чем имеется на складе, таким образом были решены. Однако при совместной работе нескольких пользователей выявились и более тонкие проблемы. В частности, таким образом можно было гарантировать правильность работы с одной таблицей, но не правильность последовательности изменений данных в нескольких таблицах (последовательность изменений данных могла быть прервана, к примеру, в середине (отключение электроэнергии, сбой при передаче по линиям связи и проч.), в результате база данных могла остаться в непредсказуемом состоянии (к примеру, товар был списан с карточки складского учета, а выписка накладной на списанный товар не была завершена)).

Именно эта проблема породила, в конце концов, понятие целостности данных, и, как следствие, понятие транзакции – группы последовательных изменений нескольких таблиц данных, которая выполняется либо «сразу» вся, либо не выполняется вовсе.

Еще одна проблема работы с данными, которая возникла при изменении стиля работы с вычислительной техникой – проблема безопасности данных. Коммерческая информация, хранящаяся в базе данных, имеет, безусловно, ценный характер. Ее утеря, повреждение, либо утечка могут иметь для владельца информации самые катастрофические последствия, поэтому вполне естественным было желание возложить обеспечение конфиденциальности информации на системы хранения данных. Способы обеспечения конфиденциальности информации достаточно разнообразны – начиная от разграничения доступа к данным и заканчивая шифрования информации, передаваемой по линиям связи.

В итоге развития средств хранения и доступа к информации сформировался определенный набор свойств, которым должна удовлетворять система хранения информации:

- модель представления данных (как правило, реляционная);
- понятие метаданных (т.е. разделение описания данных от собственно данных);
- понятие целостности данных (т.е. удовлетворение их определенным критериям);
- понятие транзакции (группа последовательных изменений данных, выполняемая либо «сразу» как единое целое, либо не выполняемая вовсе);
- средства обеспечения конфиденциальности информации.

Системы, обладающие указанным набором свойств, носят название систем управления базами данных (СУБД).

Система управления базами данных (СУБД)-Database Management System – совокупность программных и языковых средств, обеспечивающих управление базами данных (ГОСТ34.321-96 «Эталонная модель управления данными»).

Управление базами данных – процесс определения, создания, ведения баз данных, а также манипулирования ими (ГОСТ34.321-96 «Эталонная модель управления данными»)

Система управления базами данных (СУБД) – Data Base Management System (DBMS) – программная система, предназначенная для создания и хранения базы данных на основе некоторой модели данных, обеспечения логической и физической целостности содержащихся в ней данных, надежного и эффективного использования ресурсов (данных, пространства памяти и вычислительных ресурсов), предоставления к ней санкционированного доступа для приложений и конечных пользователей, а также для поддержки функций администратора базы данных. (М.Р.Когаловский, «Энциклопедия технологий баз данных»)

База данных (Data Base) – организованная в соответствии с определенными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей. (М.Р.Когаловский, «Энциклопедия технологий баз данных»)

2.2. Архитектура СУБД

Пользователей базы данных можно разделить на 3 группы: администрация БД, прикладные программисты и конечные пользователи. Естественно, каждая группа пользователей по-своему видит БД и пользуется своей терминологией при выражении своих информационных потребностей. Для обеспечения адекватного способа видения БД каждой группой ее пользователей и удовлетворения их информационных потребностей введена иерархия пользовательских представлений или уровни абстракции данных. Одной из важнейших функций СУБД является поддержание каждого из указанных уровней абстракции данных, то есть архитектура СУБД является многоуровневой. Реализация каждого уровня абстракции данных обеспечивается соответствующей функциональной компонентой СУБД. Таким образом, архитектура СУБД – это совокупность ее функциональных компонент, а также средств обеспечения их взаимодействия друг с другом и с пользователями.

В соответствие с архитектурой ANSI-SPARC предлагается трехуровневая модель архитектуры СУБД, которая поддерживает три соответствующих уровня абстракции данных – концептуальный, внутренний и внешний. Эта модель является универсальной в том смысле, что в ее рамки укладывается архитектура любой СУБД. Специфика же отдельных СУБД включает поддерживаемую модель данных и механизмы реализации взаимодействия между уровнями архитектуры СУБД.

Каждый из уровней архитектуры СУБД поддерживает соответствующие модели БД, отражающие точки зрения всех групп пользователей на одну и ту же БД. Совокупность моделей БД, соответствующих уровням абстракции данных предметной области, называется архитектурой базы данных.

Концептуальный уровень архитектуры СУБД служит для поддержания единого взгляда на БД всеми группами пользователей и общего для всех ее приложений, то есть независимо от них. Именно в среду концептуального уровня отображается инфологическая модель предметной области. Представление БД на концептуальном уровне называется концептуальной моделью БД или схемой БД. Она обеспечивает интегрированное представление о предметной области.

В СУБД для описания концептуальной модели служит язык описания данных (ЯОД). Так как, концептуальный уровень архитектуры СУБД отражает логическую структуру БД, то схема БД называется также логической моделью БД.

Внутренний уровень архитектуры СУБД поддерживает представление БД в среде хранения (на внешних ЗУ). Этот уровень называют еще физическим, т.к. именно на этом уровне база данных представлена полностью в «материализованном» виде. Описание БД на этом уровне осуществляется с помощью языка описания хранения данных (ЯОХД) СУБД и называется внутренней моделью БД. Эта модель включает описание размещения БД на внешнем ЗУ и методы доступа к данным. Во многих современных СУБД описание логической и физической моделей БД объединены.

Внешний уровень архитектуры СУБД предназначен для прикладных программистов. Описание БД на этом уровне осуществляется в СУБД с помощью языка манипулирования данными (ЯМД) СУБД и называется внешней моделью БД или подсхемой БД.

Внешняя модель БД обеспечивает реализацию конкретного приложения (задачи) и поэтому является подмножеством (проекцией) схемы БД. Количество внешних моделей БД или подсхем БД соответствует количеству приложений.

В архитектуре БД выделяют еще 4-й уровень, который не связан с СУБД и не поддерживается ею – это уровень конечного пользователя. На этом уровне поддерживается представление БД для конечных пользователей, а описание БД осуществляется на естественном языке в терминах предметной области, которыми оперируют конечные пользователи. Модель БД на этом уровне включает описание форматов сообщений и документов, составляемых в соответствии с запросами конечных пользователей и реализуемых прикладными программами.

Схема БД. Составлением схемы БД занимается администратор БД. В этой трактовке функции системного программиста БД также отнесены к прерогативам администратора БД. Тогда прикладной программист или конечный пользователь не обязательно должны знать о схеме БД, т.к. она бывает достаточно сложна. В зависимости от поддерживаемой СУБД модели данных схема БД называется реляционной, сетевой или иерархической.

Подсхемы БД. Подсхемы БД составляет администратор БД и предназначены они для прикладных программистов. Подсхема, являясь проекцией схемы БД, содержит только те отношения, и только те атрибуты этих отношений, которые используются прикладным программистом для решения конкретной задачи (запроса). При описании подсхемы БД могут быть заданы все необходимые ограничения, например: пароль доступа к отношению или его атрибутам, режим обработки отношения или его атрибутов – «только чтение» или «чтение и запись» и другие. В подсхеме БД могут также присутствовать виртуальные атрибуты – это не хранимые в БД атрибуты отношений, а вычисляемые алгоритмически для конкретной задачи (запроса). Также в подсхеме БД могут быть изменены способы упорядочивания данных (то есть, назначены другие ключи), типы данных (цифровые заменены на символьные и наоборот). При этом СУБД обеспечивает автоматическое выполнение всех этих процессов. Средства задания подсхемы БД определяются средствами поддержки внешних моделей БД в конкретной СУБД. При развитых средствах поддержки предполагается, что подсхема задается администратором (системным программистом) БД в виде рабочих. В некоторых случаях для описания подсхем БД создается язык описания данных подсхемы.

Взаимодействие прикладных программ с СУБД. Данные, размещенные в БД, доступны прикладной программе, составленной на алгоритмическом языке, не непосредственно через операторы этого языка, а с помощью языка манипулирования данными (ЯМД) СУБД. Различные СУБД содержат разный набор операторов ЯМД, однако, можно выделить группу основных операторов, имеющих аналоги в каждой СУБД:

«ВКЛЮЧИТЬ - В - БД (ЗАПОМНИТЬ, ПОМЕСТИТЬ)»;

«ИСКЛЮЧИТЬ - ИЗ - БД (УДАЛИТЬ)»;

«ИЗМЕНИТЬ»;

«ИЗВЛЕЧЬ (НАЙТИ)».

ЯМД включается в конкретный алгоритмический язык, который в этом случае называется включающим или базовым языком. Работающие по этому принципу СУБД называют СУБД с включающим или базовым языком. Практически во всех типовых СУБД ЯМД реализуется по способу базового языка. Существует два основных подхода.

В первом подходе в прикладной программе используется оператор CALL - если необходимо выполнить оператор ЯМД, то в прикладной программе средствами оператора CALL задаются требуемые действия, и осуществляется обращение к СУБД, которая после выполнения запрошенного действия возвращает управление прикладной программе.

Во втором подходе в набор операторов алгоритмического языка включаются операторы ЯМД СУБД. В прикладной программе резервируются поля, необходимые для размещения

данных, извлекаемых из БД, или подготовленных к внесению в БД. Совокупность этих полей в программе называется рабочей областью задачи (РОЗ).

Кроме того, некоторые СУБД требуют резервирования в прикладной программе памяти под буферы ввода/вывода для обмена данными между программой и БД. При этом данные сначала помещаются в буфер, а затем их подмножество, подлежащее обработке в программе, передается в РОЗ и наоборот. В некоторых СУБД с целью оптимизации режима мультидоступа (одновременный доступ к данным многих прикладных программ) буферы ввода/вывода резервируются средствами СУБД независимо от прикладных программ и одни и те же буферы могут одновременно использоваться несколькими прикладными программами.

2.2.1. СУБД и уровни описания данных

СУБД должна предоставлять доступ к данным любым пользователям, включая и тех, которые практически не имеют и (или) не хотят иметь представления: о физическом размещении в памяти данных и их описаний; о механизмах поиска запрашиваемых данных; о проблемах, возникающих при одновременном запросе одних и тех же данных многими пользователями (прикладными программами); о способах обеспечения защиты данных от некорректных обновлений и (или) несанкционированного доступа; о поддержании баз данных в актуальном состоянии и о множестве других функций СУБД.

При выполнении основных из этих функций СУБД должна использовать различные описания данных. А как создавать эти описания? Проект базы данных надо начинать с анализа предметной области и выявления требований к ней отдельных пользователей (сотрудников организации, для которых создается база данных). Подробнее этот процесс будет рассмотрен ниже, а здесь отметим, что проектирование обычно поручается человеку (группе лиц) – администратору базы данных (АБД). Им может быть как специально выделенный сотрудник организации, так и будущий пользователь базы данных, достаточно хорошо знакомый с машинной обработкой данных.

Объединяя частные представления о содержимом базы данных (внешний уровень описания данных, которому соответствует внешняя модель), полученные в результате опроса пользователей, и свои представления о данных, которые могут потребоваться в будущих приложениях, АБД сначала создает обобщенное неформальное описание создаваемой базы данных. Это описание, выполненное с использованием естественного языка, математических формул, таблиц, графиков и других средств, понятных всем людям, работающих над проектированием базы данных, называют концептуальным уровнем описания данных, которому соответствует концептуальная модель или инфологическая модель предметной области. Такая человеко-ориентированная модель полностью независима от физических параметров среды хранения данных. Поэтому концептуальная или инфологическая модель не должна изменяться до тех пор, пока какие-то изменения в реальном мире не потребуют изменения в ней некоторого определения, чтобы эта модель продолжала отражать предметную область.

Рис. 2.1. Связь программ и данных при использовании СУБД

Остальные модели, показанные на рис.2.2, являются СУБД-ориентированными. С их помощью СУБД дает возможность программам и пользователям осуществлять доступ к хранимым данным лишь по их именам, не заботясь о физическом расположении этих данных. Нужные данные отыскиваются СУБД на внешних запоминающих устройствах по физической модели данных. Так как указанный доступ осуществляется с помощью конкретной СУБД, то модели должны быть описаны на языке описания данных этой СУБД. Такое описание, создаваемое АБД по концептуальной (или инфологической) модели данных, называют логической моделью данных, поддерживаемой выбранной СУБД, или, иначе, даталогической моделью данных.

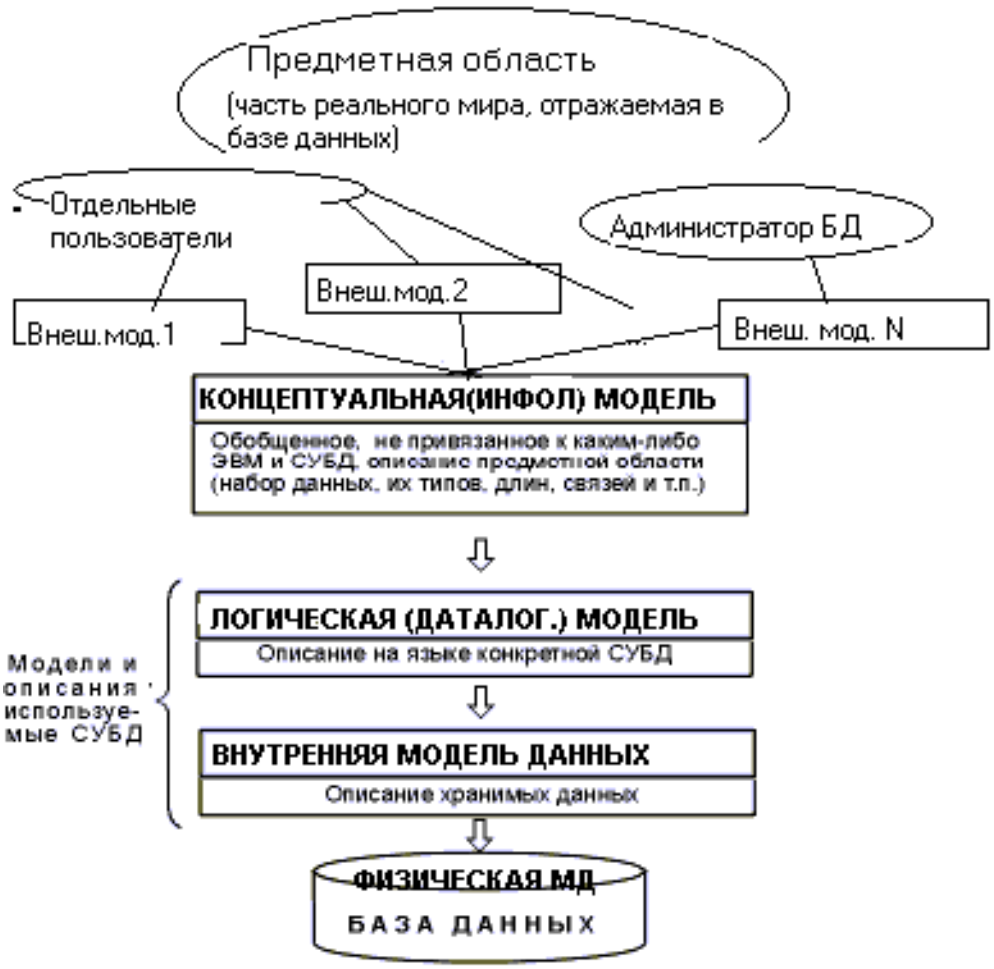

Рис. 2.2. Соответствие уровней описания данных и моделей данных

Трехуровневая архитектура позволяет обеспечить независимость хранимых данных от использующих их программ. АБД может при необходимости переписать хранимые данные на другие носители информации и (или) реорганизовать их физическую структуру, изменив лишь внутреннюю модель данных. АБД может подключить к системе любое число новых пользователей (новых приложений), дополнив, если надо, концептуальную модель. Указанные изменения внутренней и концептуальной моделей не будут замечены существующими пользователями системы (окажутся "прозрачными" для них), так же как не будут замечены и новые пользователи. Следовательно, независимость данных обеспечивает возможность развития системы баз данных без разрушения существующих приложений.

Реструктуризация и реорганизация БД. С понятием концептуальной модели БД (схемы БД) связано другое понятие – реструктуризация БД, которое означает модификацию концептуальной схемы БД с последующим приведением БД в соответствие с внесенными изменениями. Цель реструктуризации БД - упрощение схемы БД или приведение ее в соответствие с изменениями в предметной области.

С понятием внутренней модели БД связано понятие реорганизации БД, которое означает изменение структуры хранения данных или их размещения в среде хранения. Цель реорганизации БД – повышение производительности БД или предоставление освобожденной памяти на внешнем ЗУ для ее последующего использования.

2.3. Функциональные возможности СУБД

Как было сказано выше, системой управления базами данных называют программную систему, предназначенную для создания на ЭВМ общей базы данных, используемой для решения множества задач. Подобные системы служат для поддержания базы данных в актуальном состоянии и обеспечивают эффективный доступ пользователей к содержащимся в ней данным в рамках предоставленных пользователям полномочий. СУБД предназначена для централизованного управления базой данных в интересах всех работающих в этой системе. По степени универсальности различают два класса СУБД: системы общего назначения; специализированные системы.

СУБД общего назначения не ориентированы на какую-либо предметную область или на информационные потребности какой-либо группы пользователей. Каждая система такого рода реализуется как программный продукт, способный функционировать на некоторой модели ЭВМ в определенной операционной системе и поставляется многим пользователям как коммерческое изделие. Такие СУБД обладают средствами настройки на работу с конкретной базой данных. Использование СУБД общего назначения в качестве инструментального средства для создания автоматизированных информационных систем, основанных на технологии баз данных, позволяет существенно сокращать сроки разработки, экономить трудовые ресурсы. Этим СУБД присущи развитые функциональные возможности и даже определенная функциональная избыточность.

Специализированные СУБД создаются в редких случаях при невозможности или нецелесообразности использования СУБД общего назначения. СУБД общего назначения - это сложные программные комплексы, предназначенные для выполнения всей совокупности функций, связанных с созданием и эксплуатацией базы данных информационной системы. Рынок программного обеспечения ПК располагает большим числом разнообразных по своим функциональным возможностям коммерческих систем управления базами данных общего назначения, а также средствами их окружения практически для всех массовых моделей машин и для различных операционных систем.

Рассмотрим основные функциональные возможности СУБД:

1. производительность,
2. обеспечение целостности данных на уровне базы данных,
3. обеспечение безопасности,
4. работа в многопользовательских средах,
5. импорт-экспорт,
6. доступ к данным SQL,
7. возможности запросов и инструментальные средства разработки прикладных программ.

Производительность СУБД оценивается:

- временем выполнения запросов;
- скоростью поиска информации в неиндексированных полях;
- временем выполнения операций импортирования базы данных из других форматов;
- скоростью создания индексов и выполнения таких массовых операций, как обновление, вставка, удаление данных;
- максимальным числом параллельных обращений к данным в многопользовательском режиме;
- временем генерации отчета.

На производительность СУБД оказывают влияние два фактора:

1. СУБД, которые следят за соблюдением целостности данных, несут дополнительную нагрузку, которую не испытывают другие программы;

2. производительность собственных прикладных программ сильно зависит от правильного проектирования и построения базы данных.

Самые быстрые программные изделия отнюдь не обладают самыми развитыми функциональными возможностями на уровне процессора СУБД.

Обеспечение целостности данных на уровне базы данных. Эта характеристика подразумевает наличие средств, позволяющих удостовериться, что информация в базе данных всегда остается корректной и полной. Должны быть установлены правила целостности, и они должны храниться вместе с базой данных и соблюдаться на глобальном уровне. Целостность данных должна обеспечиваться независимо от того, каким образом данные заносятся в память (в интерактивном режиме, посредством импорта или с помощью специальной программы).

Обеспечение безопасности. Некоторые СУБД предусматривают средства обеспечения безопасности данных. Такие средства обеспечивают выполнение следующих операций: шифрование прикладных программ; шифрование данных; защиту паролем; ограничение уровня доступа (к базе данных, к таблице, к словарю, для пользователя).

Работа в многопользовательских средах. Практически все рассматриваемые СУБД предназначены для работы в многопользовательских средах, но обладают для этого различными возможностями. Обработка данных в многопользовательских средах предполагает выполнение программным продуктом следующих функций:

- блокировку базы данных, записи, поля;
- идентификацию станции, установившей блокировку;
- обновление информации после модификации;
- контроль за временем и повторение обращения;
- обработку транзакций (транзакция - последовательность операций пользователя над базой данных, которая сохраняет ее логическую целостность);
- работу с сетевыми системами (LAN Manager, NetWare, Unix).

Импорт-экспорт. Эта характеристика отражает: возможность обработки СУБД информации, подготовленной другими программными средствами; возможность использования другими программами данных, сформированных средствами рассматриваемой СУБД.

Доступ к данным посредством языка SQL. Язык запросов SQL (Structured Query Language) реализован в целом ряде популярных СУБД для различных типов ЭВМ либо как базовый, либо как альтернативный. В силу своего широкого использования является международным стандартом языка запросов. Язык SQL предоставляет развитые возможности как конечным пользователям, так и специалистам в области обработки данных [5].

СУБД, ориентированные на разработчиков, обладают развитыми средствами для создания приложений. К элементам инструментария разработки приложений можно отнести: мощные языки программирования; средства реализации меню, экранных форм ввода-вывода данных и генерации отчетов; средства генерации приложений (прикладных программ); генерацию исполнимых файлов. Реализация языковых интерфейсов может быть осуществлена различными способами. Для высококвалифицированных пользователей (разработчиков сложных прикладных систем) языковые средства чаще всего представляются в их явной синтаксической форме. В других случаях функции языков могут быть доступны косвенным образом, когда они реализуются в форме различного рода меню, диалоговых сценариев или заполняемых пользователем таблиц. По таким входным данным интерфейсные средства формируют адекватные синтаксические конструкции языка

интерфейса и передают их на исполнение или включают в генерируемый программный код приложения. Интерфейсы с неявным использованием языка широко используются в СУБД для персональных ЭВМ. Языковые средства используются для выполнения двух основных функций; описания представления базы данных; выполнения операций манипулирования данными. Первая из этих функций обеспечивается языком описания (определения) данных (ЯОД). ЯОД не всегда синтаксически оформляется в виде самостоятельного языка. Он может быть составной частью единого языка данных, сочетающего возможности определения данных и манипулирования данными.

Язык манипулирования данными (ЯМД) позволяет запрашивать предусмотренные в системе операции над данными из базы данных. Имеются многочисленные примеры языков СУБД, объединяющих возможности описания данных и манипулирования данными в единых синтаксических рамках. Популярным языком такого рода является реляционный язык SQL. Все рассматриваемые программные средства обладают автоматизированными средствами создания экранных форм, запросов, отчетов, меню, наклеек, стандартных писем.

2.4. Основные функции СУБД

С точки зрения пользователя СУБД представляет собой программное обеспечение, которое управляет доступом к базе данных и отвечает за реализацию следующих процессов:

- Определение данных.

СУБД должна допускать определения данных (внешние схемы, концептуальную схему, внутреннюю схему, а также все связанные отображения) в исходной форме и преобразовывать эти определения в форму соответствующих объектов. СУБД должна включать в себя компонент языкового процессора для различных языков определений данных. СУБД должна также «понимать» синтаксис языка определений данных.

- Обработка данных.

СУБД должна уметь обрабатывать запросы пользователя на выборку, изменение или удаление существующих данных в базе данных или на добавление новых данных в базу данных. СУБД должна включать в себя компонент процессора языка обработки данных.

- Безопасность и целостность данных.

СУБД должна контролировать пользовательские запросы и пресекать попытки нарушения правил безопасности и целостности, определенные АБД.

- Восстановление данных и дублирование.

СУБД или другой связанный с ней программный компонент, обычно называемый администратором транзакций, должны осуществлять необходимый контроль над восстановлением данных и дублированием.

- Словарь данных.

СУБД должна обеспечить функцию словаря данных. Словарь «содержит данные о данных» (называемые метаданными), т.е. определения других объектов системы. Исходная и объектная формы различных схем (внешних, концептуальной и т.д.) и отображений будут сохранены в словаре. Расширенный словарь будет включать также перекрестный ссылки, показывающие, например, какие из программ какую часть базы данных используют, какие отчеты требуются тем или иным пользователям, какие терминалы подключены к системе и т.д. Словарь может быть интегрирован в определяемую им базу данных, а значит, должен содержать описание самого себя. Должна быть возможность обращения к словарю, как и к другой базе данных, например, для того чтобы узнать, какие программы и/или пользователи будут затронуты при предполагаемом внесении изменения в систему.

- Производительность.

СУБД должна выполнять все указанные функции с максимально возможной производительностью.

СУБД – это пакет программ, позволяющий обеспечить пользователей языковыми средствами описания и манипулирования данными; обеспечить поддержку логических моделей данных; обеспечить операции создания и манипулирования логическими данными и одновременное отображение этих операций над физическими данными; обеспечить защиту и целостность данных, поскольку при коллективном режиме работы возможно использование общих физических данных; при этом необходимо обеспечить защиту от некорректных обновлений пользователями, защиту от несанкционированного доступа, защиту данных от разрушений при сбоях оборудования.

Перечисленные выше функции СУБД, в свою очередь, используют следующие основные функции более низкого уровня.

- Непосредственное управление данными во внешней памяти. Эта функция включает обеспечение необходимых структур внешней памяти как для хранения данных, непосредственно входящих в БД, так и для служебных целей, например, для убыстрения доступа к данным в некоторых случаях (обычно для этого используются индексы). В некоторых реализациях СУБД активно используются возможности существующих файловых систем, в других работа производится вплоть до уровня устройств внешней памяти. Но подчеркнем, что в развитых СУБД пользователи в любом случае не обязаны знать, использует ли СУБД файловую систему, и если использует, то как организованы файлы. В частности, СУБД поддерживает собственную систему именования объектов БД.

- Управление буферами оперативной памяти. СУБД обычно работают с БД значительного размера; по крайней мере, этот размер обычно существенно больше доступного объема оперативной памяти. Понятно, что если при обращении к любому элементу данных будет производиться обмен с внешней памятью, то вся система будет работать со скоростью устройства внешней памяти. Практически единственным способом реального увеличения этой скорости является буферизация данных в оперативной памяти. При этом, даже если операционная система производит общесистемную буферизацию (как в случае ОС UNIX), этого недостаточно для целей СУБД, которая располагает гораздо большей информацией о полезности буферизации той или иной части БД. Поэтому в развитых СУБД поддерживается собственный набор буферов оперативной памяти с собственной дисциплиной замены буферов. Заметим, что существует отдельное направление СУБД, которое ориентировано на постоянное присутствие в оперативной памяти всей БД. Это направление основывается на предположении, что в будущем объем оперативной памяти компьютеров будет настолько велик, что позволит не беспокоиться о буферизации. Пока эти работы находятся в стадии исследований.

- Управление транзакциями. Транзакция - это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется, и СУБД фиксирует (COMMIT) изменения БД, произведенные этой транзакцией, во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД. Понятие транзакции необходимо для поддержания логической целостности БД. Если вспомнить наш пример информационной системы с файлами СОТРУДНИКИ и ОТДЕЛЫ, то единственным способом не нарушить целостность БД при выполнении операции приема на работу нового сотрудника является объединение элементарных операций над файлами СОТРУДНИКИ и ОТДЕЛЫ в одну транзакцию. Таким образом, поддержание механизма транзакций является обязательным условием даже однопользовательских СУБД (если, конечно, такая система заслуживает названия СУБД). Но понятие транзакции гораздо более важно в многопользовательских СУБД. То свойство, что каждая транзакция начинается при целостном состоянии БД и оставляет это состояние целостным после своего завершения, делает очень удобным использование понятия транзакции как единицы активности пользователя по отношению к БД. При соответствующем управлении параллельно

выполняющимися транзакциями со стороны СУБД каждый из пользователей может в принципе ощущать себя единственным пользователем СУБД (на самом деле, это несколько идеализированное представление, поскольку в некоторых случаях пользователи многопользовательских СУБД могут ощутить присутствие своих коллег).

С управлением транзакциями в многопользовательской СУБД связаны важные понятия сериализации транзакций и сериального плана выполнения смеси транзакций. Под сериализаций параллельно выполняющихся транзакций понимается такой порядок планирования их работы, при котором суммарный эффект смеси транзакций эквивалентен эффекту их некоторого последовательного выполнения. Сериальный план выполнения смеси транзакций - это такой план, который приводит к сериализации транзакций. Понятно, что если удается добиться действительно сериального выполнения смеси транзакций, то для каждого пользователя, по инициативе которого образована транзакция, присутствие других транзакций будет незаметно (если не считать некоторого замедления работы по сравнению с однопользовательским режимом).

Существует несколько базовых алгоритмов сериализации транзакций. В централизованных СУБД наиболее распространены алгоритмы, основанные на синхронизационных захватах объектов БД. При использовании любого алгоритма сериализации возможны ситуации конфликтов между двумя или более транзакциями по доступу к объектам БД. В этом случае для поддержания сериализации необходимо выполнить откат (ликвидировать все изменения, произведенные в БД) одной или более транзакций. Это один из случаев, когда пользователь многопользовательской СУБД может реально (и достаточно неприятно) ощутить присутствие в системе транзакций других пользователей.

- Журнализация. Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя. Обычно рассматриваются два возможных вида аппаратных сбоев: так называемые мягкие сбои, которые можно трактовать как внезапную остановку работы компьютера (например, аварийное выключение питания), и жесткие сбои, характеризуемые потерей информации на носителях внешней памяти. Примерами программных сбоев могут быть: аварийное завершение работы СУБД (по причине ошибки в программе или в результате некоторого аппаратного сбоя) или аварийное завершение пользовательской программы, в результате чего некоторая транзакция остается незавершенной. Первую ситуацию можно рассматривать как особый вид мягкого аппаратного сбоя; при возникновении последней требуется ликвидировать последствия только одной транзакции. Понятно, что в любом случае для восстановления БД нужно располагать некоторой дополнительной информацией. Другими словами, поддержание надежности хранения данных в БД требует избыточности хранения данных, причем та часть данных, которая используется для восстановления, должна храниться особо надежно. Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД.

Журнал - это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью (иногда поддерживаются две копии журнала, располагаемые на разных физических дисках), в которую поступают записи обо всех изменениях основной части БД. В разных СУБД изменения БД журнализируются на разных уровнях: иногда запись в журнале соответствует некоторой логической операции изменения БД (например, операции удаления строки из таблицы реляционной БД), иногда - минимальной внутренней операции модификации страницы внешней памяти; в некоторых системах одновременно используются оба подхода.

Во всех случаях придерживаются стратегии "упреждающей" записи в журнал (так называемого протокола Write Ahead Log - WAL). Грубо говоря, эта стратегия заключается в

том, что запись об изменении любого объекта БД должна попасть во внешнюю память журнала раньше, чем измененный объект попадет во внешнюю память основной части БД. Известно, что если в СУБД корректно соблюдается протокол WAL, то с помощью журнала можно решить все проблемы восстановления БД после любого сбоя.

Самая простая ситуация восстановления - индивидуальный откат транзакции. Строго говоря, для этого не требуется общесистемный журнал изменений БД. Достаточно для каждой транзакции поддерживать локальный журнал операций модификации БД, выполненных в этой транзакции, и производить откат транзакции путем выполнения обратных операций, следуя от конца локального журнала. В некоторых СУБД так и делают, но в большинстве систем локальные журналы не поддерживают, а индивидуальный откат транзакции выполняют по общесистемному журналу, для чего все записи от одной транзакции связывают обратным списком (от конца к началу). При мягком сбое во внешней памяти основной части БД могут находиться объекты, модифицированные транзакциями, не закончившимися к моменту сбоя, и могут отсутствовать объекты, модифицированные транзакциями, которые к моменту сбоя успешно завершились (по причине использования буферов оперативной памяти, содержимое которых при мягком сбое пропадает). При соблюдении протокола WAL во внешней памяти журнала должны гарантированно находиться записи, относящиеся к операциям модификации обоих видов объектов. Целью процесса восстановления после мягкого сбоя является состояние внешней памяти основной части БД, которое возникло бы при фиксации во внешней памяти изменений всех завершившихся транзакций и которое не содержало бы никаких следов незаконченных транзакций. Для того, чтобы этого добиться, сначала производят откат незавершенных транзакций (undo), а потом повторно воспроизводят (redo) те операции завершенных транзакций, результаты которых не отображены во внешней памяти. Этот процесс содержит много тонкостей, связанных с общей организацией управления буферами и журналом. Более подробно мы рассмотрим это в соответствующей лекции.

Для восстановления БД после жесткого сбоя используют журнал и архивную копию БД. Грубо говоря, архивная копия - это полная копия БД к моменту начала заполнения журнала (имеется много вариантов более гибкой трактовки смысла архивной копии). Конечно, для нормального восстановления БД после жесткого сбоя необходимо, чтобы журнал не пропал. Как уже отмечалось, к сохранности журнала во внешней памяти в СУБД предъявляются особо повышенные требования. Тогда восстановление БД состоит в том, что исходя из архивной копии по журналу воспроизводится работа всех транзакций, которые закончились к моменту сбоя. В принципе, можно даже воспроизвести работу незавершенных транзакций и продолжить их работу после завершения восстановления. Однако в реальных системах это обычно не делается, поскольку процесс восстановления после жесткого сбоя является достаточно длительным.

- Поддержка языков БД. Для работы с базами данных используются специальные языки, в целом называемые языками баз данных. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков. Чаще всего выделялись два языка - язык определения схемы БД (SDL - Schema Definition Language) и язык манипулирования данными (DML - Data Manipulation Language). SDL служил главным образом для определения логической структуры БД, т.е. той структуры БД, какой она представляется пользователям. DML содержал набор операторов манипулирования данными, т.е. операторов, позволяющих заносить данные в БД, удалять, модифицировать или выбирать существующие данные. В современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс с базами данных. Стандартным языком наиболее распространенных в настоящее время реляционных СУБД является язык SQL (Structured Query Language). В нескольких лекциях этого курса язык SQL будет рассматриваться достаточно подробно, а пока мы перечислим основные функции реляционной СУБД, поддерживаемые на "языковом" уровне (т.е. функции, поддерживаемые

при реализации интерфейса SQL). Прежде всего, язык SQL сочетает средства SDL и DML, т.е. позволяет определять схему реляционной БД и манипулировать данными. При этом именование объектов БД (для реляционной БД - именование таблиц и их столбцов) поддерживается на языковом уровне в том смысле, что компилятор языка SQL производит преобразование имен объектов в их внутренние идентификаторы на основании специально поддерживаемых служебных таблиц-каталогов. Внутренняя часть СУБД (ядро) вообще не работает с именами таблиц и их столбцов.

Язык SQL содержит специальные средства определения ограничений целостности БД. Опять же, ограничения целостности хранятся в специальных таблицах-каталогах, и обеспечение контроля целостности БД производится на языковом уровне, т.е. при компиляции операторов модификации БД компилятор SQL на основании имеющихся в БД ограничений целостности генерирует соответствующий программный код.

Специальные операторы языка SQL позволяют определять так называемые представления БД, фактически являющиеся хранимыми в БД запросами (результатом любого запроса к реляционной БД является таблица) с именованными столбцами. Для пользователя представление является такой же таблицей, как любая базовая таблица, хранимая в БД, но с помощью представлений можно ограничить или наоборот расширить видимость БД для конкретного пользователя. Поддержание представлений производится также на языковом уровне.

Наконец, авторизация доступа к объектам БД производится также на основе специального набора операторов SQL. Идея состоит в том, что для выполнения операторов SQL разного вида пользователь должен обладать различными полномочиями. Пользователь, создавший таблицу БД, обладает полным набором полномочий для работы с этой таблицей. В число этих полномочий входит полномочие на передачу всех или части полномочий другим пользователям, включая полномочие на передачу полномочий. Полномочия пользователей описываются в специальных таблицах-каталогах, контроль полномочий поддерживается на языковом уровне.

2.5. Типовая организация современной СУБД

Естественно, организация типичной СУБД и состав ее компонентов соответствует рассмотренному нами набору функций. Напомним, что мы выделили следующие основные функции СУБД: управление данными во внешней памяти; управление буферами оперативной памяти; управление транзакциями; журнализация и восстановление БД после сбоев; поддержание языков БД.

Логически в современной реляционной СУБД можно выделить наиболее внутреннюю часть - ядро СУБД (часто его называют Data Base Engine), компилятор языка БД (обычно SQL), подсистему поддержки времени выполнения, набор утилит. В некоторых системах эти части выделяются явно, в других - нет, но логически такое разделение можно провести во всех СУБД.

Ядро СУБД отвечает за управление данными во внешней памяти, управление буферами оперативной памяти, управление транзакциями и журнализацию. Соответственно, можно выделить такие компоненты ядра (по крайней мере, логически, хотя в некоторых системах эти компоненты выделяются явно), как менеджер данных, менеджер буферов, менеджер транзакций и менеджер журнала. Как можно было понять из первой части этой лекции, функции этих компонентов взаимосвязаны, и для обеспечения корректной работы СУБД все эти компоненты должны взаимодействовать по тщательно продуманным и проверенным протоколам. Ядро СУБД обладает собственным интерфейсом, не доступным пользователям напрямую и используемым в программах, производимых компилятором SQL (или в подсистеме поддержки выполнения таких программ) и утилитах БД. Ядро СУБД является

основной резидентной частью СУБД. При использовании архитектуры "клиент-сервер" ядро является основной составляющей серверной части системы.

Основной функцией компилятора языка БД является компиляция операторов языка БД в некоторую выполняемую программу. Основной проблемой реляционных СУБД является то, что языки этих систем (а это, как правило, SQL) являются непроцедурными, т.е. в операторе такого языка специфицируется некоторое действие над БД, но эта спецификация не является процедурой, а лишь описывает в некоторой форме условия совершения желаемого действия (вспомните примеры из первой лекции). Поэтому компилятор должен решить, каким образом выполнять оператор языка прежде, чем произвести программу. Применяются достаточно сложные методы оптимизации операторов, которые мы подробно рассмотрим в следующих лекциях. Результатом компиляции является выполняемая программа, представляемая в некоторых системах в машинных кодах, но более часто в выполняемом внутреннем машинно-независимом коде. В последнем случае реальное выполнение оператора производится с привлечением подсистемы поддержки времени выполнения, представляющей собой, по сути дела, интерпретатор этого внутреннего языка.

Наконец, в отдельные утилиты БД обычно выделяют такие процедуры, которые слишком накладно выполнять с использованием языка БД, например, загрузка и выгрузка БД, сбор статистики, глобальная проверка целостности БД и т.д. Утилиты программируются с использованием интерфейса ядра СУБД, а иногда даже с проникновением внутрь ядра.

2.6. Характеристика основных типов систем управления базами данных

На раннем этапе развития СУБД данные обрабатывались и хранились с использованием наиболее популярных в те времена типов компьютеров - мэйнфреймов семейства IBM-360/370 (их отечественные аналоги серии ЕС, производившиеся странами СЭВ) и мини-ЭВМ типа DEC PDP-11 (у которых также был отечественный аналог - СМ-4/СМ-1420). Как правило, при работе с такими компьютерами использовались неинтеллектуальные терминалы, управляемые все тем же мэйнфреймом или мини-ЭВМ.

Надо сказать, обработка данных с помощью мэйнфреймов и мини-ЭВМ имела свои преимущества, в определенной степени утраченные позже, в эпоху персональных компьютеров и настольных СУБД. К ним, в частности, относились:

- возможность коллективного использования ресурсов и оборудования, например центрального процессора, оперативной памяти, внешних устройств (принтеров, плоттеров, накопителей на магнитной ленте и иных устройств хранения данных и т.д.);
- централизованное хранение данных.

Серьезным недостатком подобных систем было практическое отсутствие персонализации рабочей среды - все программное обеспечение, включая текстовые редакторы, компиляторы, СУБД, хранилось также централизованно и использовалось коллективно.

Этот недостаток был одной из причин бурного роста индустрии персональных компьютеров - наряду с простотой в эксплуатации и невысокой стоимостью по сравнению с мэйнфреймами и мини-ЭВМ пользователей привлекали возможности персонализации рабочей среды, в особенности возможность выбора наиболее подходящего данному пользователю программного обеспечения. Именно в тот период и начался бурный рост популярности настольных СУБД, таких как dBase (РЕБУС) и, чуть позже, FoxBASE, Paradox, а также некоторых других, ныне благополучно забытых. Надо сказать, в то время происходили процессы заимствования и стандартизации удачных идей и подходов, что особенно заметно отразилось на судьбе такого продукта, как dBase, чей язык

программирования и принципы организации данных были заимствованы многими другими производителями в своих продуктах.

2.6.1. Настольные СУБД

Настольные СУБД как таковые не содержат специальных приложений и сервисов, управляющих данными, - взаимодействие с ними осуществляется с помощью файловых сервисов операционной системы. Нередко подобные СУБД имеют в своем составе и средства разработки, ориентированные на работу с данными формата, характерного для этой СУБД, и позволяющие создать более или менее комфортный пользовательский интерфейс. Что же касается обработки данных - она целиком и полностью осуществляется в пользовательском (клиентском) приложении.

Следующим шагом в развитии настольных СУБД было появление их сетевых многопользовательских версий, позволяющих обрабатывать данные, находящиеся в общедоступном хранилище (например, на сетевом диске) нескольким пользователям одновременно. От чисто настольных СУБД их многопользовательские версии отличаются наличием механизма блокировок частей файлов данных (содержащих одну или несколько записей таблицы), что позволяет обращаться к одному и тому же файлу нескольким пользователям одновременно.

Недостатки подобных СУБД не очевидны и становятся заметны, как правило, при росте хранимых объемов данных и увеличении числа пользователей. Обычно они проявляются в снижении производительности и в возникновении сбоев при обработке данных после некоторого времени использования клиентских приложений. Причина подобных проблем кроется в основном принципе работы таких СУБД и основанных на них информационных систем, заключающемся в обработке данных внутри пользовательского приложения. Например, если с помощью такой системы требуется выполнить запрос согласно какому-либо критерию (например, выбрать заказы, обработанные за последние два часа, из таблицы заказов), то, в лучшем случае (если эта таблица проиндексирована по времени поступления заказа), приложение должно прочесть с сетевого диска весь индекс, найти в нем сведения о местоположении записей в файлах, содержащих таблицу, и затем прочесть эти части файлов. В общем же случае, когда таблица не проиндексирована по данному полю, ее необходимо загрузить с сетевого диска и проанализировать.

Еще одна проблема настольных СУБД заключается в возможности нарушения ссылочной целостности данных, так как единственным механизмом, контролирующим ее, является пользовательское приложение. Поэтому все пользовательские приложения должны содержать соответствующий код и доступ к файлам базы данных из любых других приложений должен быть запрещен. В наиболее популярных настольных СУБД (например, Microsoft Access, Corel Paradox) код, контролирующий стандартную ссылочную целостность, содержится в библиотеках, используемых всеми приложениями, работающими с этой базой данных, а сама база данных при этом может содержать описание правил ссылочной целостности.

Наиболее популярные настольные СУБД. На сегодняшний день известно более двух десятков форматов данных настольных СУБД, однако наиболее популярными, исходя из числа проданных копий, следует признать dBase, Paradox, FoxPro и Access. Из появившихся недавно СУБД следует также отметить Microsoft Data Engine - по существу серверную СУБД, представляющую собой <облегченную> версию Microsoft SQL Server, но предназначенную, тем не менее, для использования главным образом в настольных системах и небольших рабочих группах.

Сведения о производителях перечисленных выше СУБД представлены в табл. 2.1.

Таблица 2.1

Фирмы-производители СУБД

СУБД	Производитель	URL
Visual dBase	DBase.Inc	http://www.dbase2000.com/
Paradox	Corel	http://www.corel.com/
Microsoft Access 2000	Microsoft	http://www.microsoft.com/
Microsoft FoxPro	Microsoft	http://www.microsoft.com/
Microsoft Visual FoxPro	Microsoft	http://www.microsoft.com/
Microsoft Visual FoxPro	Microsoft	http://www.microsoft.com/
Microsoft Data Engine	Microsoft	http://www.microsoft.com/

dBase и Visual dBase

Первая промышленная версия СУБД dBase - dBase II (принадлежащая тогда компании Ashton-Tate, приобретенной позже компанией Borland) появилась в начале 80-х годов. Благодаря простоте в использовании, нетребовательности к ресурсам компьютера и, что не менее важно, грамотной маркетинговой политике компании-производителя этот продукт приобрел немалую популярность, а с выходом следующих его версий - dBase III и dBase III Plus (1986 г.), оснащенных весьма комфортной по тем временам средой разработки и средствами манипуляции данными, быстро занял лидирующие позиции среди настольных СУБД и средств создания использующих их приложений.

Хранение данных в dBase основано на принципе «одна таблица - один файл» (эти файлы обычно имеют расширение *.dbf). MEMO-поля и BLOB-поля (доступные в поздних версиях dBase) хранятся в отдельных файлах (обычно с расширением *.dbt). Индексы для таблиц также хранятся в отдельных файлах. При этом в ранних версиях этой СУБД требовалась специальная операция реиндексирования для приведения индексов в соответствие с текущим состоянием таблицы.

Формат данных dBase является открытым, что позволило ряду других производителей заимствовать его для создания dBase-подобных СУБД, частично совместимых с dBase по форматам данных. Например, весьма популярная некогда СУБД FoxBase (разработанная Fox Software, Inc. и ныне принадлежащая Microsoft) использовала формат данных dBase для таблиц, однако форматы для хранения MEMO-полей и индексов были своими собственными, несовместимыми с dBase. Очень популярное в начале 90-х годов (и кое-где применяемое до сих пор) средство разработки Clipper компании Nantucket Corp (приобретенной впоследствии компанией Computer Associates) манипулировало как с данными формата dBase III (включая индексные файлы и файлы для MEMO-полей), так и с индексными файлами собственного формата.

Помимо популярного формата данных dBase является родоначальником и некогда популярного семейства языков программирования, получившего называние xBase. Все языки этого семейства, использующиеся и в FoxBase, и в Clipper, и в некоторых более поздних средствах разработки, таких как канувший в Лету CA Visual Objects фирмы Computer Associates, содержат сходный набор команд для манипуляции данными и являются по существу интерпретируемыми языками. В роли интерпретатора команд xBase выступает обычно либо среда разработки приложения на этом языке, либо среда времени выполнения, которую можно поставлять вместе с приложением. Отметим, что для скрытия исходного

текста xBase-приложения подобные СУБД обычно содержат утилиты для псевдокомпиляции кода, который затем поставляется вместе со средой времени выполнения. В случае Clipper среда времени выполнения содержится в самом исполняемом файле (и сам Clipper формально считается компилятором), но тем не менее этот язык по существу также является интерпретируемым.

Для работы с данными формата dBase (или иных dBase-подобных СУБД) совершенно необязательно пользоваться диалектами xBase. Доступ к этим данным возможен с помощью ODBC API (и соответствующих драйверов) и некоторых других механизмов доступа к данным (например, Borland Database Engine, некоторых библиотек других производителей типа CodeBase фирмы Sequenter), и это позволяет создавать приложения, использующие формат данных dBase, практически с помощью любого средства разработки, поддерживающего один из этих механизмов доступа к данным.

После покупки dBase компанией Borland этот продукт, получивший впоследствии название Visual dBase, приобрел набор дополнительных возможностей, характерных для средств разработки этой компании и для имевшейся у нее другой настольной СУБД - Paradox. Среди этих возможностей были специальные типы полей для графических данных, поддерживаемые индексы, хранение правил ссылочной целостности внутри самой базы данных, а также возможность манипулировать данными других форматов, в частности серверных СУБД, за счет использования BDE API и SQL Links.

В настоящее время Visual dBase принадлежит компании dBase.Inc. Его последняя версия имеет следующие возможности:

- Средства манипуляции данными dBase и FoxPro всех версий.
- Средства создания форм, отчетов и приложений.
- Средства публикации данных в Internet и создания Web-клиентов.
- Ядро доступа к данным Advantage Database Server фирмы Extended Systems и ODBC-драйвер для доступа к данным этой СУБД.
- Средства публикации отчетов в Web.
- Средства визуального построения запросов.
- Средства генерации исполняемых файлов и дистрибутивов.

В настоящее время к Visual dBase в качестве дополнения может быть приобретен компонент dConnections, позволяющий осуществить доступ к данным Oracle, Sybase, Informix, MS SQL Server, DB2, InterBase из Visual dBase 7.5 и приложений, созданных с его помощью. Компания dBase.Inc объявила также о проекте dBASE Open Source, целью которого является разработка сообществом пользователей dBase новых компонентов и классов с целью включения их в последующую версию dBase (получившую название dBase 2000).

Paradox

Paradox был разработан компанией Ansa Software, и первая его версия увидела свет в 1985 году. Этот продукт был впоследствии приобретен компанией Borland. С июля 1996 года он принадлежит компании Corel и является составной частью Corel Office Professional.

В конце 80-х - начале 90-х годов Paradox, принадлежавший тогда компании Borland International, был весьма популярной СУБД, в том числе и в нашей стране, где он одно время занимал устойчивые позиции на рынке средств разработки настольных приложений с базами данных.

Принцип хранения данных в Paradox сходен с принципами хранения данных в dBase - каждая таблица хранится в своем файле (расширение *.db), MEMO- и BLOB-поля хранятся в отдельном файле (расширение *.md), как и индексы (расширение *.px).

Однако, в отличие от dBase, формат данных Paradox не является открытым, поэтому для доступа к данным этого формата требуются специальные библиотеки. Например, в

приложениях, написанных на C или Pascal, использовалась некогда популярная библиотека Paradox Engine, ставшая основой Borland Database Engine. Эта библиотека используется ныне в приложениях, созданных с помощью средств разработки Borland (Delphi, C++Builder), в некоторых генераторах отчетов (например, Crystal Reports) и в самом Paradox. Существуют и ODBC-драйверы к базам данных, созданным различными версиями этой СУБД.

Отметим, однако, что отсутствие «открытости» формата данных имеет и свои достоинства. Так как в этой ситуации доступ к данным осуществляется только с помощью «знающих» этот формат библиотек, простое редактирование подобных данных по сравнению с данными открытых форматов типа dBase существенно затруднено. В этом случае возможны такие недоступные при использовании «открытых» форматов данных сервисы, как защита таблиц и отдельных полей паролем, хранение некоторых правил ссылочной целостности в самих таблицах - все эти сервисы предоставляются Paradox, начиная с первых версий этой СУБД.

По сравнению с аналогичными версиями dBase ранние версии Paradox обычно предоставляли разработчикам баз данных существенно более расширенные возможности, такие как использование деловой графики в DOS-приложениях, обновление данных в приложениях при многопользовательской работе, визуальные средства построения запросов, на основе интерфейса QBE - Query by Example (запрос по образцу), средства статистического анализа данных, а также средства визуального построения интерфейсов пользовательских приложений с автоматической генерацией кода на языке программирования PAL (Paradox Application Language).

Windows-версии СУБД Paradox помимо перечисленных выше сервисов позволяли также манипулировать данными других форматов, в частности dBase и данными, хранящимися в серверных СУБД. Такую возможность пользователи Paradox получили благодаря использованию библиотеки Borland Database Engine и драйверов SQL Links. Это позволило использовать Paradox в качестве универсального средства управления различными базами данных (существенно облегченная версия Paradox 7 под названием Database Desktop по-прежнему входит в состав Borland Delphi и Borland C++Builder именно с этой целью). Что же касается базового формата данных, используемого в этом продукте, то он обладает теми же недостатками, что и все форматы данных настольных СУБД, и поэтому при возможности его стараются заменить на серверную СУБД, даже сохранив сам Paradox как средство разработки приложений и манипуляции данными.

Текущие версии данной СУБД содержат:

- Средства манипуляции данными Paradox и dBase.
- Средства создания форм, отчетов и приложений.
- Средства визуального построения запросов.
- Средства публикации данных и отчетов в Internet и создания Web-клиентов.
- Corel Web-сервер.
- ODBC-драйвер для доступа к данным формата Paradox из Windows-приложений.
- Средства для доступа к данным формата Paradox из Java-приложений.
- Run-time-версию Paradox для поставки вместе с приложениями.
- Средства создания дистрибутивов.
- Драйверы SQL Links для доступа к данным серверных СУБД.

Microsoft FoxPro и Visual FoxPro

FoxPro ведет свое происхождение от настольной СУБД FoxBase фирмы Fox Software. Разрабатывая FoxBase в конце 80-х годов, эта компания преследовала цель создать СУБД, функционально совместимую с dBase с точки зрения организации файлов и языка программирования, но существенно превышающую ее по производительности. Одним из способов повышения производительности являлась более эффективная организация

индексных файлов, нежели в dBase, - по формату индексных файлов эти две СУБД несовместимы между собой.

По сравнению с аналогичными версиями dBase FoxBase и более поздняя версия этого продукта, получившая название FoxPro, предоставляли своим пользователям несколько более широкие возможности, такие как использование деловой графики, генерация кода приложений, автоматическая генерация документации к приложениям и т.д.

Впоследствии этот продукт был приобретен компанией Microsoft. Его последние версии (начиная с версии 3.0, выпущенной в 1995 году) получили название Visual FoxPro. С каждой новой версией этот продукт оказывался все более и более интегрирован с другими продуктами Microsoft, в частности с Microsoft SQL Server, - в состав Visual FoxPro в течение нескольких последних лет входят средства переноса данных FoxPro в SQL Server и средства доступа к данным этого сервера из Visual FoxPro и созданных с его помощью приложений. Хотя формат данных FoxPro также модифицировался с каждой новой версией, приобретая такие возможности, как хранение правил ссылочной целостности и некоторых бизнес-правил в самой базе данных, миграции приложений Visual FoxPro на серверные платформы уделялось значительно большее внимание.

Последняя версия этого продукта доступна и отдельно, и как составная часть Microsoft Visual Studio. Отличительной особенностью этой настольной СУБД от двух, рассмотренных выше, является интеграция этого продукта с технологиями Microsoft, в частности, поддержка COM (Component Object Model - компонентная объектная модель, являющаяся основой функционирования 32-разрядных версий Windows и организации распределенных вычислений в этой операционной системе), интеграция с Microsoft SQL Server, возможности создания распределенных приложений, основанных на концепции Windows DNA (Distributed interNet Applications).

Visual FoxPro предоставляет следующие возможности:

- Средства публикации данных в Internet и создания Web-клиентов.
- Средства создания ASP-компонентов и Web-приложений.
- Средства создания COM-объектов и объектов для Microsoft Transaction Server, позволяющих создавать масштабируемые многозвенные приложения для обработки данных.
- Средства доступа к данным серверных СУБД, базирующиеся на использовании OLE DB (набор COM-интерфейсов, позволяющий осуществить унифицированный доступ к данным из разнообразных источников, в том числе из нереляционных баз данных и иных источников, например Microsoft Exchange).
- Средства доступа к данным Microsoft SQL Server и Oracle, включая возможность создания и редактирования таблиц, триггеров, хранимых процедур
- Средства отладки хранимых процедур Microsoft SQL Server.
- Средство визуального моделирования компонентов и объектов, являющиеся составными частями приложения - Visual Modeller.
- Средство для управления компонентами приложений, позволяющее осуществлять их повторное использование.

Тенденции развития этого продукта очевидны: из настольной СУБД Visual FoxPro постепенно превращается в средство разработки приложений в архитектуре «клиент-сервер» и распределенных приложений в архитектуре Windows DNA. Впрочем, эти тенденции в определенной степени характерны для всех наиболее популярных настольных СУБД - мы уже убедились, что и dBase, и Paradox также позволяют осуществлять доступ к наиболее популярным серверным СУБД.

Microsoft Access

Первая версия СУБД Access появилась в начале 90-х годов. Это была первая настольная реляционная СУБД для 16-разрядной версии Windows. Популярность Access значительно возросла после включения этой СУБД в состав Microsoft Office.

В отличие от Visual FoxPro, фактически превратившегося в средство разработки приложений, Access ориентирован в первую очередь на пользователей Microsoft Office, в том числе и не знакомых с программированием. Это, в частности, проявилось в том, что вся информация, относящаяся к конкретной базе данных, а именно таблицы, индексы (естественно, поддерживаемые), правила ссылочной целостности, бизнес-правила, список пользователей, а также формы и отчеты хранятся в одном файле, что в целом удобно для начинающих пользователей.

Последняя версия этой СУБД - Access 2000 входит в состав Microsoft Office 2000 Professional и Premium, а также доступна как самостоятельный продукт. В состав Access 2000 входят:

- Средства манипуляции данными Access и данными, доступными через ODBC (последние могут быть присоединены к базе данных Access).
- Средства создания форм, отчетов и приложений; при этом отчеты могут быть экспортированы в формат Microsoft Word или Microsoft Excel, а для создания приложений используется Visual Basic for Applications, общий для всех составных частей Microsoft Office.
- Средства публикации отчетов в Internet.
- Средства создания интерактивных Web-приложений для работы с данными (Data Access Pages).
- Средства доступа к данным серверных СУБД через OLE DB.
- Средства создания клиентских приложений для Microsoft SQL Server.
- Средства администрирования Microsoft SQL Server.

Поддержка COM в Access выражается в возможности использовать элементы управления ActiveX в формах и Web-страницах, созданных с помощью Access. В отличие от Visual FoxPro создание COM-серверов с помощью Access не предполагается, т.е. Microsoft Access может быть использован, с одной стороны, в качестве настольной СУБД и составной части офисного пакета, а с другой стороны, в качестве клиента Microsoft SQL Server, позволяющего осуществлять его администрирование, манипуляцию его данными и создание приложений для этого сервера.

Помимо манипуляции данными Microsoft SQL Server, Access 2000 позволяет также в качестве хранилища данных использовать Microsoft Data Engine (MSDE), представляющий собой по существу настольный сервер баз данных, совместимый с Microsoft SQL Server.

Microsoft Data Engine

MSDE представляет собой СУБД, базирующуюся на технологиях Microsoft SQL Server, но предназначенную для использования в настольных системах или в сетевых приложениях с объемом данных до 2 Гбайт и небольшим количеством пользователей. По существу MSDE является облегченной версией Microsoft SQL Server, не содержащей средств администрирования, и к настольным СУБД может быть отнесена весьма условно.

В Microsoft Access пользователь может выбрать механизм доступа к данным: Microsoft Jet - стандартный набор библиотек доступа к данным или MSDE (в этом случае управление базой данных осуществляется с помощью отдельного процесса). Возможно преобразование имеющихся баз данных Access в базу данных MSDE из среды разработки Access.

Базы данных MSDE полностью совместимы с базами данных Microsoft SQL Server и могут при необходимости управляться этим сервером. Как большинство серверных СУБД, эти базы данных поддерживают транзакции, позволяют создавать триггеры и хранимые процедуры (недоступные в базах данных Access), использовать механизмы защиты данных,

предоставляемые операционной системой. Помимо этого при большом числе пользователей и большом объеме данных приложения, использующие MSDE, отличаются более высокой производительностью, так как обработка запросов происходит внутри процесса, управляющего базой данных, а не внутри клиентского приложения, что позволяет снизить сетевой трафик, связанный с передачей данных от сервера к клиенту.

MSDE входит в состав Microsoft Office 2000 Premium или Developer, а также доступна на Web-сайте Microsoft для зарегистрированных пользователей Visual Studio 6.0 Professional, Enterprise Edition либо любого из средств разработки, являющегося частью Visual Studio 6.0 Professional или Enterprise Edition. MSDE может свободно распространяться в составе приложений, созданных с помощью любого из средств разработки, входящего в состав Visual Studio 6.0 или Office 2000 Developer.

Мы рассмотрели наиболее популярные на сегодняшний день настольные СУБД и проследили историю их развития. Мы увидели, что развитие тех из настольных СУБД, что сумели сохранить свою популярность на протяжении многих лет, подчинялось вполне определенным закономерностям. Все эти СУБД:

- приобрели визуальные средства проектирования форм, отчетов и приложений в момент появления ранних Windows-версий;
- стали предоставлять доступ к данным серверных СУБД к моменту появления первых 32-разрядных версий;
- приобрели средства публикации данных в Internet и в той или иной степени поддерживают создание приложений для редактирования данных с помощью Web-браузеров;
- начали предоставлять возможность хранить описания правил ссылочной целостности внутри базы данных.

Помимо этого все современные СУБД, за исключением Corel Paradox, в качестве альтернативы собственному формату данных позволяют использовать для создания настольных приложений облегченные серверы баз данных, предназначенные для использования на одном компьютере или в рамках небольшой рабочей группы. Иными словами, история развития настольных СУБД отражает современные тенденции развития информационных систем, такие как создание распределенных систем с использованием Internet или Intranet, применение средств быстрой разработки приложений и массовый перенос приложений, использующих базы данных, включая настольные приложения, в архитектуру «клиент-сервер».

2.6.2. Серверные СУБД

Радикальным решением проблемы сетевого трафика и иных проблем, возникающих при увеличении объема данных и числа пользователей, стал переход к архитектуре «клиент-сервер», позаимствовавшей многие достоинства старой «мэйнфреймовой» модели вычислений, в частности централизацию хранения и обработки данных.

Принцип централизации хранения и обработки данных является базовым принципом архитектуры «клиент-сервер». Для его реализации используется так называемый сервер баз данных, выполненный как приложение или сервис операционной системы, и только он может реально манипулировать файлами, в которых хранятся данные, - для клиентских приложений эти файлы абсолютно бесполезны (и, при правильной организации доступа пользователей к файлам в сети, вообще не должны быть доступны).

Сервер баз данных осуществляет целый комплекс действий по управлению данными. Основными его обязанностями являются:

- выполнение пользовательских запросов на выбор и модификацию данных и метаданных, получаемых от клиентских приложений, функционирующих на персональных компьютерах локальной сети;
- хранение и резервное копирование данных;
- поддержка ссылочной целостности данных согласно определенным в базе данных правилам;
- обеспечение авторизованного доступа к данным на основе проверки прав и привилегий пользователей;
- протоколирование операций и ведение журнала транзакций.

В качестве рабочего места пользователя может быть использован обычный персональный компьютер, что позволяет не отказываться от привычной рабочей среды. В простейшем случае клиент-серверная информационная система состоит из двух основных компонентов:

1. сервера баз данных, управляющего данными и выполняющего запросы клиентских приложений;
2. клиентских приложений, предоставляющих интерфейс пользователя и посылающих запросы к серверу.

Существуют и более сложные реализации архитектуры «клиент-сервер», например многозвенные информационные системы с использованием серверов приложений, реализующих бизнес-логику и осуществляющих обработку данных. Однако обсуждение архитектуры таких систем находится за пределами данного пособия.

Преимущества архитектуры «клиент-сервер». Информационные системы, использующие архитектуру «клиент-сервер», обладают серьезными преимуществами по сравнению с их аналогами, созданными на основе сетевых версий настольных СУБД. Рассмотрим наиболее важные из них.

Одним из важнейших преимуществ архитектуры «клиент-сервер» является снижение сетевого трафика при выполнении запросов. Например, при необходимости выбора пяти записей из таблицы, содержащей миллион записей, клиентское приложение посылает серверу запрос, который сервером анализируется на корректность и, если запрос корректен, компилируется, оптимизируется и выполняется. После этого результат запроса (те самые пять записей, а вовсе не вся таблица) передается обратно клиенту. При этом, формулируя запрос, можно не задумываться о том, есть ли в базе данных индексы, способные облегчить поиск нужных записей, - если они есть, то они будут использованы сервером, а если нет, запрос все равно будет выполнен, хотя, возможно, это займет больше времени, чем при использовании индексов. Но в любом случае - есть индексы или нет - в клиентское приложение передается только результат запроса, и в этом случае сетевой трафик не зависит ни от их наличия, ни от числа записей в таблицах, к которым произведен запрос.

Вторым преимуществом архитектуры «клиент-сервер» является возможность хранения бизнес-правил (например, правил ссылочной целостности или ограничений на значения данных) на сервере, что позволяет избежать дублирования кода в различных клиентских приложениях, использующих общую базу данных. Кроме того, в этом случае любое редактирование данных, в том числе и редактирование средствами, не предусмотренными разработчиками информационной системы (например, различными утилитами администрирования сервера), может быть произведено только с учетом этих правил. Если последние версии некоторых настольных СУБД и способны хранить ограничения на значения данных, либо тексты SQL-запросов, триггеры и хранимые процедуры в них, как правило, отсутствуют - их просто некому выполнять. Исключением здесь, пожалуй, является только Microsoft Data Engine, но, отнести к настольным эту СУБД можно весьма условно - фактически MSDE представляет собой локальный сервер баз данных, обладающий всеми характерными для серверной СУБД атрибутами, включая отдельный серверный процесс.

Для описания серверных бизнес-правил в наиболее типичных ситуациях (например, при реализации связей master/detail) нередко используются так называемые CASE-средства (CASE означает Computer-Aided System Engineering) для создания диаграмм «сущность-связь». Эти инструменты позволяют описать подобные правила на уровне логической и физической моделей данных без какого бы то ни было программирования, а затем сгенерировать код соответствующих серверных объектов - триггеров, хранимых процедур, серверных ограничений. В этом случае клиентские приложения будут избавлены от значительной части кода, связанного с реализацией бизнес-правил непосредственно в приложении. Часть кода, связанного с обработкой данных, также может быть реализована в виде хранимых процедур сервера, что позволяет еще больше «облегчить» клиентское приложение, а это означает, что требования к рабочим станциям могут быть не столь высоки. Это, в конечном итоге, снижает стоимость информационной системы даже при использовании дорогостоящих серверной СУБД и аппаратного обеспечения.

Помимо перечисленных выше трех преимуществ следует также отметить, что современные серверные СУБД обладают широкими возможностями управления пользовательскими привилегиями и правами доступа к различным объектам базы данных. Как правило, в базе данных хранятся сведения о ее пользователях, их паролях и привилегиях, а каждый объект базы данных принадлежит какому-либо пользователю. Владелец объекта может предоставить другим пользователям право использовать его тем или иным способом (например, позволить читать из него данные какому-либо другому пользователю). Большинство серверных СУБД поддерживает так называемые роли, представляющие собой совокупность прав на доступ к тем или иным объектам базы данных. В этом случае каждый пользователь может иметь одну или несколько ролей и соответственно определенные в этих ролях привилегии.

Современные серверные СУБД обладают также широкими возможностями резервного копирования и архивации данных, а нередко и оптимизации выполнения запросов. Они, также как правило, предоставляют возможность параллельной обработки данных, особенно в случае использования многопроцессорных компьютеров в качестве сервера баз данных.

Рассмотрим какими способами решается проблема модернизации устаревших информационных систем, основанных на настольных СУБД, с целью перехода к архитектуре «клиент-сервер», и с какими проблемами можно при этом столкнуться.

Модернизация устаревших информационных систем. С проблемой модернизации устаревших информационных систем рано или поздно сталкивается почти каждый разработчик или IT-менеджер. В нашей стране все еще нетрудно встретить банковские системы, использующие dBase, а также относительно свежие коммерческие разработки, созданные с помощью Clipper, средством обмена данными которым служат не Internet и не сетевые протоколы, а курьер с дискетами и электричка (это вовсе не всегда происходит из-за неосведомленности разработчиков - просто у их клиентов нет и в ближайшее время не будет денег на приличное оборудование и соответствующую инфраструктуру). Именно поэтому мы полагаем, что проблема модернизации информационных систем в России еще долго будет оставаться актуальной.

В каждой организации, переживающей процесс модернизации информационной системы, возникают свои проблемы. В одной пользователи требуют сохранить привычный пользовательский интерфейс; в другой нужно суметь не просто перенести в новую базу унаследованные данные, но и изменить их в соответствии с вновь возникшими потребностями (например, исправить ошибки, допущенные много лет назад при проектировании данных, или объединить данные из нескольких разных источников); в третьей организации сохранилось и используется большое количество отчетов, созданных с помощью старой настольной СУБД, и нужно обеспечить возможность их использования в новой информационной системе; в четвертой процесс ввода новых данных происходит

непрерывно, и это накладывает ограничения на то, как именно производить процесс замены СУБД и клиентских приложений.

В целом варианты модернизации информационной системы можно условно разделить на следующие категории.

1. Замена СУБД с сохранением структуры базы данных и пользовательских приложений (или относительно небольшой их модернизацией).

2. Замена и СУБД, и пользовательских приложений с сохранением структуры базы данных.

3. Замена СУБД, пользовательских приложений и одновременная модернизация структуры базы данных.

Если говорить о первом варианте модернизации, в этом отношении больше всего повезло пользователям Microsoft Access - процесс замены базы данных Microsoft Access на MSDE (или Microsoft SQL Server) происходит достаточно безболезненно, и пользовательские приложения при этом обычно сохраняют свою работоспособность. Так как в данном случае все эти СУБД (Access, MSDE и SQL Server) принадлежат одному производителю, перенос данных между ними осуществляется вполне корректно, с сохранением всех определенных в базе данных бизнес-правил. Кроме того, утилиты переноса данных из Access содержатся в комплекте поставки и других серверных СУБД (например, Oracle). Относительно безболезненно можно осуществить перенос данных из Visual FoxPro в Microsoft SQL Server.

Что касается замены других версий настольных СУБД на серверные, здесь могут возникнуть определенные проблемы. Например, при переносе данных из dBase или Paradox в какую-нибудь серверную СУБД обслуживающее их приложение, написанное на Delphi, вполне может отказаться работать даже после корректной перенастройки библиотек доступа к данным, особенно если оно содержит сведения о метаданных. Возможны также различные неприятности, связанные с использованием строчных и прописных букв в наименованиях полей, применением русских наименований полей (и вообще локализованных версий, поддерживающих национальные традиции написания чисел и дат и нередко превращающих числовые данные и даты в при переносе в другую СУБД в нечто невообразимое), несовместимости некоторых типов данных (это особенно часто происходит при переносе таблиц Paradox в другие СУБД).

Переход к архитектуре «клиент-сервер» не исчерпывается переносом данных в новую СУБД. Чтобы воспользоваться всеми преимуществами этой архитектуры, следует также реализовать бизнес-правила, содержащиеся в клиентском приложении, в виде триггеров и хранимых процедур, а затем модифицировать код клиентского приложения, удалив реализацию бизнес-правил или заменив ее на обращения к соответствующим объектам базы данных.

Второй вариант модернизации информационной системы представляет собой по существу создание нового проекта по готовой модели данных плюс только что рассмотренный перенос данных в новую СУБД.

Третий вариант модернизации можно рассматривать как два самостоятельных проекта. Первый из них представляет собой создание информационной системы практически «с нуля», второй - заполнение базы унаследованными данными. При этом, поскольку структуры баз данных различны, стандартными утилитами переноса данных (имеющимися в комплектах поставки многих средств разработки и серверных СУБД) здесь обычно обойтись не удается - как правило, в этом случае создаются одноразовые приложения, производящие нужные преобразования данных в процессе их переноса.

2.6.3. Характерные черты современных серверных СУБД

Выше мы обсудили преимущества архитектуры «клиент-сервер», обусловленные ее базовыми принципами и не зависящие от того, какая конкретно СУБД выбрана. Однако помимо собственно средства современные серверные СУБД обычно предоставляют дополнительный набор сервисов, связанных с обслуживанием хранения и обработки данных, созданием клиентских приложений, сменой СУБД или ее версии, обслуживанием нескольких баз данных, публикацией данных в Internet. Поскольку в условиях конкуренции между различными производителями СУБД каждый из них стремится к завоеванию как можно большей части рынка, это приводит к тому, что все они пытаются перегнать друг друга и предоставить потенциальному потребителю (а также производителям средств разработки, средств проектирования данных и генераторов отчетов) как можно больше сервисов различного назначения, так как при выборе СУБД потребитель обычно ориентируется на то, что поддерживает данная СУБД, с одной стороны, и чем она поддерживается - с другой. И, как и в случае настольных СУБД, в этой области также происходит процесс заимствования идей, решений и интерфейсов.

Рассмотрим некоторые сервисы, предлагаемые СУБД.

Реализация для нескольких платформ. Почти все современные серверы баз данных существуют в нескольких версиях для различных платформ (как правило, различные коммерческие версии UNIX - Solaris, HP/UX и др., а также Windows NT Server и, с недавнего времени, Windows 2000). Многие производители серверных СУБД также выпускают версии своих серверов для Windows NT Workstation и Windows 2000 и далее (а иногда даже для Windows CE). В последнем случае такие серверы нередко бывают персональными (однопользовательскими), либо в их реализации отсутствуют возможности, характерные для полнофункциональных версий этих серверов. Подобные персональные серверы, как правило, используются в «переносных» системах, например на ноутбуках, применяемых для работы с базой данных отдельно от центрального офиса компании с последующей репликацией данных. Иногда такие серверы используются при создании приложений для изоляции разработчика от сервера баз данных, находящегося в процессе эксплуатации. В последнее время многие производители серверных СУБД выпускают также версии для Linux.

Исключением из этого правила является Microsoft SQL Server. Для данной СУБД это вполне оправданно - компания Microsoft сама производит серверные операционные системы и в отличие от большинства других производителей серверных СУБД может себе позволить создавать серверы баз данных, тесно интегрированные с сервисами операционной системы собственного производства и поддерживающие исключительно их.

Административные утилиты. Администрирование сервера баз данных, конечно, удел профессионалов. Однако и профессионал предпочтет удобные утилиты администрирования унылому окну с интерфейсом командной строки. Наличие удобных утилит администрирования, как ни странно, иногда оказывается одним из решающих факторов при выборе СУБД. Именно поэтому подавляющее большинство современных СУБД обычно поставляется с подобными утилитами, и их интерфейс в последнее время напоминает интерфейс Windows Explorer .

Следует отметить, что производители СУБД, вовремя не заметившие эту тенденцию, оказались лишенными значительной части рынка или практически вытесненными с него. Весьма показателен здесь пример IB Database (InterBase Corporation) - производители этой СУБД, очень неплохой с точки зрения хранения данных и скорости обработки запросов, не позаботились вовремя об удобных утилитах администрирования (а также о средствах репликации данных), положившись на продукты третьих фирм и на продвижение этой СУБД в составе популярных средств разработки, - и, в результате, многие ее потенциальные пользователи сделали выбор в пользу других СУБД, а сама IB Database практически перешла в разряд некоммерческих продуктов с доступными исходными текстами.

Резервное копирование данных. Резервное копирование данных и журналов транзакций поддерживается всеми без исключения коммерческими серверными СУБД. Различия между СУБД в поддержке резервного копирования заключаются в том, возможно ли производить резервное копирование в процессе работы пользователей, и если да, то какие пользовательские операции в это время нельзя выполнять. Помимо этого в комплекте поставки некоторых СУБД могут содержаться утилиты для использования различных внешних устройств (например, накопителей на магнитных лентах) в качестве средств хранения резервных копий.

Обслуживание репликаций. Репликация по существу представляет собой гарантированное копирование информации из одной базы в несколько других. Репликации используются для разделения нагрузки между серверами в сети, для перемещения поднаборов данных на вспомогательные серверы, для синхронизации данных на нескольких серверах и многих других целей.

В том или ином виде репликации поддерживаются всеми современными серверными СУБД. Различия могут быть лишь в поддержке тех или иных конкретных сценариев репликаций (например, внесение изменений одновременно на нескольких серверах, возможность шифрования реплицируемых данных и др.). Исключение здесь составляет IB Database, в текущей версии не поддерживающая репликаций, но об этой СУБД мы уже упоминали.

Если вы планируете иметь несколько серверов баз данных с необходимостью синхронизации данных (например, у вашего предприятия существует несколько филиалов), стоит обратить внимание на то, какие сценарии репликаций поддерживаются в выбранной вами СУБД.

Параллельная обработка данных в многопроцессорных системах. О возможностях повышения производительности с помощью параллельной обработки запросов в многопроцессорных системах начали говорить несколько лет назад, после появления первого продукта такого класса - Oracle Parallel Server. Серверы, поддерживающие параллельную обработку, разрешают нескольким процессорам обращаться к одной базе данных, что позволяет обеспечить высокую скорость обработки транзакций.

В настоящее время подавляющее большинство производителей современных серверных СУБД поставляют на рынок версии, поддерживающие параллельную обработку данных. Обычно это версии для Windows NT и коммерческих версий UNIX. Если вы рассчитываете использовать многопроцессорный компьютер в качестве сервера баз данных, вам имеет смысл обратить внимание на то, поддерживает ли выбранная вами СУБД параллельную обработку данных.

Поддержка OLAP и создания хранилищ данных. OLAP (On-Line Analytical Processing) представляет собой технологию построения многомерных хранилищ данных (Data Warehouses), как правило, агрегатных, то есть являющихся результатом обработки набора данных, нередко состоящего из нескольких таблиц. Такие хранилища данных в последнее время широко используются в системах поддержки принятия решений. Более подробно об идеях, лежащих в основе OLAP, будет рассказано в одной из последующих статей данного цикла, здесь же мы ограничимся лишь упоминанием этой возможности.

Многомерные хранилища данных могут быть реализованы как в виде набора обычных реляционных таблиц, так и в виде нереляционной многомерной базы данных. В последнем случае такое хранилище обычно управляется отдельным сервером. Многие производители серверных СУБД поставляют такие серверы отдельно (Oracle, Informix), некоторые включают их в состав сервера реляционных баз данных (Microsoft SQL Server 7.0). Нередко с целью повышения конкурентоспособности подобные OLAP-системы строят многомерные хранилища на основе данных из других СУБД, как это сделано, например, в Microsoft SQL Server OLAP Extensions и в Sybase Adaptive Server IQ. Если вы планируете создавать системы поддержки принятия решений, вам необходимо обратить внимание на то, какие OLAP-

серверы можно использовать на выбранной вами платформе, и поддерживают ли они выбранную вами СУБД.

Распределенные запросы и транзакции. О распределенных транзакциях и запросах заговорили в последнее время, когда наличие нескольких серверов баз данных в одной организации стало обычным явлением. Нужно отметить, что возможности выполнения распределенного запроса или распределенной транзакции поддерживаются сейчас почти всеми серверными СУБД, по крайней мере в том случае, когда все вовлеченные в транзакцию серверы - от одного и того же производителя. С этой целью используется механизм двухфазного завершения транзакций (two-phase commit), когда на первом этапе серверы, вовлеченные в транзакцию, сигнализируют о готовности ее завершить, а на втором этапе происходит реальная фиксация изменений в базах данных. Что касается распределенных транзакций с участием <чужих> серверов, то они, как правило, реализуются с помощью мониторов транзакций или иных подобных сервисов, например Microsoft Distributed Transaction Coordinator.

Если вы планируете использовать несколько серверов баз данных и совершать операции, затрагивающие данные на нескольких серверах, есть смысл поинтересоваться, поддерживают ли выбранные вами СУБД двухфазное завершение транзакций.

Средства проектирования данных. Многие производители серверных СУБД производят также средства анализа бизнес-процессов и проектирования данных, иногда универсальные (как в случае Sybase DataArchitect), а порой ориентированные главным образом на конкретную СУБД (как в случае Oracle Designer/2000). Многие производители СУБД не имеют в своем арсенале собственных средств проектирования данных, ориентируясь на универсальные CASE-средства типа Platinum ERwin. Нередко производители СУБД встраивают в административные утилиты несложные средства проектирования данных, позволяющие визуально редактировать схемы данных, как это сделано, например, в Microsoft SQL Server 7.0. Выбирая CASE-средство, обратите внимание на то, поддерживает ли оно выбранную вами СУБД.

Поддержка средств разработки и генераторов отчетов. Многие производители серверных СУБД выпускают также средства разработки и генераторы отчетов. Иногда данные средства разработки используют тот же язык программирования, что применяется при написании триггеров и хранимых процедур (в этом случае, как правило, клиентское приложение должно включать интерпретатор этого языка), что позволяет отлаживать хранимые процедуры, помещая их в клиентское приложение. Типичный пример подобного подхода реализован в Oracle Developer/2000. Однако чаще средства разработки производителей серверных СУБД используют языки программирования, отличные от языков создания серверного кода (характерный пример - четыре средства разработки Microsoft).

Практически все производители серверных СУБД делают все возможное для того, чтобы клиентские приложения для их СУБД можно было создавать с помощью других средств разработки. С этой целью они предоставляют разработчикам описания API клиентской части, ODBC-драйверы, OLE DB-провайдеры, а нередко и объектные модели, позволяя использовать COM-объекты клиентской части в приложениях (как, например, это сделано в клиентских частях Oracle, Microsoft SQL Server, Informix).

Выбирая СУБД и средство разработки, обратите внимание на то, какие механизмы доступа к данным при этом можно использовать и какие именно серверные объекты в этом случае будут доступны клиентскому приложению.

Поддержка доступа к данным с помощью Internet. Без поддержки публикации данных в Internet или получения данных от удаленных Internet-клиентов сегодня не обходится практически ни одна коммерческая СУБД, в том числе настольные базы данных. Тем или иным способом производители серверных СУБД поддерживают Web-технологии. Чаще всего эта поддержка осуществляется с помощью Web-серверов собственного производства, либо посредством создания расширений для существующих Web-серверов, либо просто

путем включения в комплект поставки утилит, генерирующих Web-страницы согласно определенному расписанию.

2.6.4. Наиболее популярные серверные СУБД

На сегодняшний день известно более двух десятков серверных СУБД, однако наиболее популярными, исходя из числа продаж и инсталляций, следует признать Oracle, Microsoft SQL Server, Informix, Sybase, DB2. Сведения о производителях перечисленных выше СУБД представлены в табл. 2.2.

Таблица 2.2

Производители серверных СУБД

СУБД	Производитель	Url
Oracle	Oracle Corp.	http://www.oracle.com/
Microsoft SQL Server	Microsoft	http://www.microsoft.com/
Informix	Informix	http://www.informix.com/
Sybase	Sybase	http://www.sybase.com/
DB2	IBM	http://www-4.ibm.com/

Oracle

Oracle была первой коммерческой реляционной СУБД, поддерживающей ставший ныне индустриальным стандартом язык SQL; ее первая версия появилась в 1979 году. Фактически все это время Oracle является бессменным лидером на рынке производителей коммерческих СУБД и второй (после Microsoft) по величине компанией, производящей программное обеспечение.

Ранние версии этой СУБД были предназначены для мэйнфреймов, а в качестве рабочих мест использовались «неинтеллектуальные» терминалы. Однако со временем появились версии Oracle, предназначенные для использования в архитектуре «клиент-сервер» (первой такой версией была Oracle 5, выпущенная в 1985 году). Первоначально эти версии были предназначены для различных серверных платформ - различных версий UNIX, VMS и др. Позже были выпущены версии сервера Oracle для Novell NetWare. Первые версии этого сервера для персональных компьютеров появились в середине 90-х (Personal Oracle 7 for Windows 3.1, Personal Oracle 7 for Windows 95, Personal Oracle Lite, Oracle Workgroup Server 7 for Windows NT). До появления этих версий персональные компьютеры могли использоваться исключительно в качестве клиентских рабочих станций - в состав Oracle для серверных платформ обычно входила клиентская часть для DOS.

Oracle была первой компанией, создавшей СУБД, использовавшую предоставляемые некоторыми серверными платформами средства параллельных вычислений - Oracle Parallel Server (до его появления параллельные вычисления использовались только для решения научных задач). При использовании параллельных вычислений Oracle Parallel Server дает возможность нескольким процессорам обращаться к одной базе данных, что позволяет обеспечить высокую скорость обработки транзакций, а более поздние его версии дают возможность осуществить декомпозицию операций с большими объемами данных с целью параллельного выполнения их на нескольких процессорах.

Отличительными свойствами версии Oracle8i являются:

- наличие объектных расширений и соответствующих типов данных, таких как вложенные таблицы, массивы, объекты и др. Иными словами, Oracle8 и Oracle8i являются объектно-ориентированными СУБД;
- наличие функций аналитической обработки данных (например, вычисления процентных соотношений, ранжирования, сравнения временных периодов);
- возможность создания таблиц, содержащих агрегатные данные (materialized views) и возможность частичного их обновления при изменении данных, на основании которых они вычислены;
- поддержка Java, в частности JDK 1.2 и JDBC 2.0;
- поддержка XML, в частности в Oracle8i включены XML Parser for Java, C/C++, PL/SQL, превращающие XML-данные в вид, пригодный для использования в Oracle8;
- поддержка HTML- и XML-страниц с включенным в них кодом PL/SQL (для их выполнения требуются дополнительные продукты, например WebDB PL/SQL Gateway или Oracle Application Server PL/SQL Cartridge);
- поддержка хранения мультимедиа-данных с возможностью индексации, построения контекстных запросов, поддержки разных языков для хранимых документов;
- набор процедур и функций для обработки пространственной информации (Oracle Spatial);
- дополнительные возможности обеспечения безопасности, например шифрование данных, поддержка SSL, использование ролей уровня базы данных и уровня предприятия;
- возможность создания систем, устойчивых к сбоям, с использованием нескольких параллельных процессов;
- поддержка Microsoft Cluster Server;
- наличие OLE DB-провайдера для доступа к данным.

Oracle 8i существует в трех редакциях: Oracle 8i, Oracle 8i Enterprise Edition, Oracle 8i Personal Edition.

Для создания многомерных хранилищ данных существует и отдельный продукт - Oracle Express OLAP.

Помимо различных версий сервера баз данных среди продуктов Oracle имеется также Designer/2000 - ориентированное на эту СУБД CASE-средство для анализа бизнес-процессов и проектирования данных, а также средства разработки клиентских приложений. Одно из них - Developer/2000 (называвшееся ранее Oracle*Forms) - весьма популярно среди пользователей Oracle; были и другие средства разработки (например, Oracle Power Objects). Отметим, что приложения, созданные с помощью Developer/2000, могут выполняться на различных платформах. Язык PL/SQL, используемый в этом средстве разработки, является интерпретируемым и представляет собой тот же самый язык, что используется в Oracle для написания серверного кода. Это позволяет отлаживать с помощью Developer/2000 серверный код.

Производя собственные средства разработки, Oracle предоставляет своим пользователям возможность создавать клиентские приложения с помощью других средств. В частности, помимо стандартного в таких случаях клиентского API (Oracle Call Interface) клиентская часть Oracle содержит также объектную модель (Oracle Objects for OLE), позволяющую использовать клиентскую часть Oracle как набор COM-объектов для доступа к данным. Кроме того, обычно клиентская часть Oracle содержит также ODBC-драйвер для доступа к данным этой СУБД.

Многие другие компании производят ODBC-драйверы и OLE DB-провайдеры для доступа к Oracle (в частности, Microsoft). Компании, производящие средства разработки, использующие собственные библиотеки доступа к данным (такие как Inprise или

Gupta/Centura), также включают библиотеки доступа к Oracle в состав наиболее дорогих версий своих продуктов.

Из готовых информационных систем на базе Oracle следует особо отметить несколько крупных систем управления предприятием, в частности SAP/R3. На Западе также нередко используются готовые решения от самой Oracle Corporation, объединенные под общим названием Oracle Applications, такие как Oracle Financials, Oracle Human Resources, Oracle Market Management, Oracle Project Systems и др.

Microsoft SQL Server

Первая версия Microsoft SQL Server, совместно разработанная в 1988 году компаниями Microsoft и Sybase, предназначалась для платформы OS/2. Последующие версии этого сервера баз данных предназначались для платформы Windows NT и со временем были тесно интегрированы с этой операционной системой. Для других платформ версии этого сервера не выпускались и не выпускаются.

Удобный пользовательский интерфейс утилит администрирования в сочетании с достаточно высокой производительностью и относительно невысокой стоимостью эксплуатации сделал эту серверную СУБД второй по популярности - после Oracle. Наибольший рост популярности этой СУБД пришелся на конец 90-х годов, когда были выпущены Microsoft SQL Server 6.0 (1995 год), обладавший централизованными функциями администрирования и встроенными возможностями репликации данных, Microsoft SQL Server 6.5 (1996 год) и Microsoft SQL Server 6.5 Enterprise Edition, поддерживающий параллельные вычисления в многопроцессорных системах.

На сегодняшний день наиболее широко используемой является выпущенная в 1998 году версия Microsoft SQL Server 7.0. Эта версия отличается от предыдущих тем, что была полностью переписана фирмой Microsoft исключительно под платформу Windows NT. В состав Microsoft SQL Server 7.0 входят еще более простые утилиты администрирования (Enterprise Manager), сервисы преобразования данных (Data Transformation Services), облегчающие перенос данных в SQL Server из других типов СУБД, поддержка распределенных запросов и транзакций, OLAP-сервер и утилиты для создания хранилищ данных (в том числе данных из других серверных СУБД), расширенная поддержкой функций для создания Web-приложений.

Помимо собственно Microsoft SQL Server 7.0 в качестве встроенной СУБД для настольных приложений и приложений для небольших рабочих групп можно также использовать Microsoft Data Engine (MSDE) - настольный сервер баз данных, совместимый с Microsoft SQL Server и предназначенный для использования в настольных системах или в сетевых приложениях с небольшим (до 2 Гбайт) объемом данных и небольшим количеством пользователей. Базы данных MSDE полностью совместимы с базами данных Microsoft SQL Server и могут при необходимости управляться этим сервером. Клиентские приложения для Microsoft SQL Server и MSDE можно создавать как с помощью средств разработки Microsoft - Visual Basic, Visual C++, Access и Visual FoxPro, так и с помощью средств разработки других производителей. Для этой цели имеются ODBC-драйвер и OLE DB-провайдер, а также содержащий их набор библиотек Microsoft Data Access Components (MDAC), позволяющий использовать в средствах разработки объекты ActiveX Data Objects (ADO) - COM-объекты для доступа к данным. MDAC является составной частью Windows 2000, а для пользователей других Windows-платформ доступен отдельно на Web-сайте Microsoft.

Sybase

Серверные продукты компании Sybase происходят от двух «предков». Первым из них является одна из ранних версий Microsoft SQL Server, созданная совместно Microsoft и Sybase. Начиная с 1994 года Microsoft и Sybase разрабатывают свои серверные продукты независимо друг от друга, и результатом деятельности компании Sybase в этом направлении является продукт под названием Adaptive Server Enterprise (в настоящее время используются его версии 11 и 12). Этот продукт существует для Windows NT и некоторых версий UNIX

(включая Linux) и предназначен для обслуживания крупных предприятий. В настоящее время этот сервер поддерживает:

- упреждающее асинхронное чтение, что повышает скорость выполнения сложных запросов;
- использование кластерных систем;
- распределенную обработку запросов, в том числе к базам данных других производителей;
- расширенные хранимые процедуры, позволяющие осуществить легкий доступ к не-SQL функциям (Java, 3GL-системы, функции операционной системы и т.д.);
- параллельную обработку запросов в многопроцессорных системах;
- параллельную работу утилит администрирования;
- интеграцию с популярными системами безопасности, такими как Kerberos.

Еще одна линия серверных продуктов Sybase ведет свое начало от сервера баз данных Watcom SQL Anywhere, отличавшегося компактностью и простотой администрирования. Последняя версия этого продукта называется Adaptive Server Anywhere 6.0.3. Этот сервер предназначен для обслуживания небольших рабочих групп, для применения в портативных компьютерах в качестве персонального сервера с периодической репликацией, а также в мобильных устройствах - существуют версии этого сервера для Windows CE 2.1 и версия UltraLite для разнообразных мобильных устройств (исключая разве что тамагочи).

Для управления распределенными транзакциями Sybase выпускает монитор транзакций - Jaguar CTS.

Для создания многомерных хранилищ данных у Sybase существует еще один серверный продукт - Adaptive Server IQ, позволяющий создавать хранилища на основе данных не только из СУБД производства Sybase, но и из СУБД других производителей. Существует ряд продуктов под общим названием Sybase Industry Warehouse Studio, ориентированных на обслуживание конкретных предметных областей: торговли (Retail Warehouse Studio), здравоохранения (Healthcare Warehouse Studio), страхования (Life Insurance Warehouse Studio) и др. Помимо серверных продуктов Sybase производит средства разработки, ориентированные на создание клиентских приложений для них (PowerBuilder, PowerJ, PowerSite; последнее предназначено для создания Web-приложений), и средства проектирования данных и генерации кода приложений. Последние можно отнести к универсальным средствам - CASE-средство DataArchitect поддерживает широкий спектр СУБД различных производителей, а генератор приложений AppModeler способен генерировать код не только для PowerBuilder и Optima++, но и для Delphi, Visual Basic, Web-приложений с использованием ASP.

Informix

Ведущий продукт фирмы Informix - Informix Dynamic Server, одной из последних версий которого являетсяя Informix Dynamic Server.2000 (выпущена в сентябре 1999 года). Данный продукт поддерживает платформы UNIX и Microsoft Windows NT и обеспечивает эффективную работу как на одно-, так и на многопроцессорных системах, а также в кластерах. Сервер построен по архитектуре Dynamic Scalable Architecture (DSA), обеспечивающей мощные средства для параллельной обработки данных. В числе основных характеристик Informix Dynamic Server следует отметить:

- использование для управления дисковым пространством как средств операционной системы (UNIX или Microsoft Windows NT), так и собственных функций, позволяющих обойти ограничения операционной системы и добиться более высокой производительности, - такое управление дисковым пространством называется Raw Disk Management;
- управление разделением памяти - поддержку одновременного доступа к данным, находящимся в памяти, несколькими приложениями;

- динамическое управление потоками;
- поддержку фрагментации таблиц и индексов на нескольких дисках;
- распараллеливание запросов (parallel database query, PDQ);
- зеркалирование данных.

Сервер поддерживает двухфазное завершение транзакций, гетерогенные транзакции (в этом случае в транзакциях может принимать участие и не-Informix сервер, доступный через Informix Enterprise Gateway).

Расширения функциональности сервера реализуются на базе DataBlade - коллекций объектов баз данных и подпрограмм на языке C, подключаемых к базе данных. Для разработки DataBlades необходимо использовать DataBlade Developer's Kit. Фирма Informix и целый ряд независимых производителей выпускают модули DataBlade, такие, например, как Excalibur Text DataBlade Module, Informix Geodetic DataBlade Module, Informix TimeSeries DataBlade Module, Excalibur Image DataBlade Module, Informix Web DataBlade Module и ряд других.

Входящие в состав Informix Dynamic Server клиентские утилиты предназначены для подключения к серверу и обработки информации (DB-Access) и для выполнения функций администрирования (DB/Cockpit).

Клиентские приложения могут создаваться с использованием языков Informix ESQL (средство для разработки на языке C, позволяющее включать в приложения запросы к данным на языке SQL), а также C, C++, Java, Visual Basic и Delphi. Помимо этого существует собственное средство разработки - Informix-4GL и Informix Client Software Developer's Kit.

Фирма Informix выпускает Informix ODBC Driver, OLE DB Provider для Informix Dynamic Server и Informix JDBC Driver.

В состав продукта входят собственно сервер, а также Informix Connect 2.30, DataBlade Developer's Kit 4.0 и Informix Server Administrator 1.0.

Для генерации отчетов предлагается Informix-ViewPoint - визуальное средство, рассчитанное на пользователей. Версия Pro также содержит средства администрирования.

Говоря о сервере фирмы Informix, следует упомянуть и поддержку OLAP: продукт под названием Informix MetaCube, который поставляется как часть Informix Decision Frontier - комплексного решения для создания хранилищ данных.

Среди других продуктов фирмы Informix следует отметить:

- Informix Internet Foundation.2000 - специально разработанный для Internet вариант Informix Dynamic Server;
- i.Reach - корпоративный репозитарий для хранения данных различного типа, интеллектуального управления информацией и извлечения данных. Основное назначение данного продукта - поддержка включения в содержимое корпоративных сайтов электронных документов и их последующее обслуживание;
- i.Sell - комплексное решение для электронной коммерции на базе Informix Dynamic Server.

DB2

Семейство серверных СУБД фирмы IBM, известное под названием DB2 Universal Database, представляет собой стратегию IBM по объединению продуктов DB2 для различных платформ в единую линию. Впервые появившееся в 1996 году семейство DB2 Universal Database объединяло в себе функциональные возможности таких продуктов фирмы, как DB2 Common Server, DB2 Parallel Edition (DB2 PE), Net.Data, Data Propagator и технологии DataHub, и предназначалось для платформ UNIX, OS/2 и Microsoft Windows NT.

При переносе DB2 на не-IBM-платформы фирма старается максимально использовать уникальные функциональные возможности конкретной платформы. Например, в DB2 for

Windows 2000 для обеспечения безопасности используется Windows NT LAN Manager, полностью поддерживается Windows Performance Monitor, Systems Management Server, интеграция с Active Directory для каталогизации баз данных, а также такие интерфейсы доступа к данным, как ODBC, ADO и OLE DB. Помимо этого DB2 for Windows 2000 поддерживает Microsoft Transaction Services (MTS) в качестве координатора при создании приложений, использующих распределенные транзакции.

Для разработчиков, использующих Microsoft Visual Studio, становятся доступными дополнительные модули, например Stored ProcedureBuilder, включаемый непосредственно в среду Visual Studio. IBM также предлагает собственные средства разработки, например IBM VisualAge for Java, позволяющие создавать приложения, работающие с данными в DB2. Продукт также поддерживает создание хранимых процедур на языке Java (Java Stored Procedure Builder).

IBM предлагает бесплатное средство для миграции данных из Microsoft Access в DB2, а также средства для миграции данных из Oracle, Microsoft, Sybase и Informix.

К основным характеристикам СУБД можно отнести: поддержку реляционных и комплексных данных через объектные расширения, возможность работы на мультипроцессорных платформах, поддержку кластеров, 64-битную архитектуру памяти и распараллеливание запросов, возможность создания Web-приложений (поддерживаются такие технологии, как Java, JDBC, SQLJ, XML) и наличие средства для гетерогенного администрирования и обработки данных.

Семейство DB2 функционионирует на системах AS/400 и RISC System/6000, мэйнфреймах IBM, машинах от Hewlett-Packard и Sun Microsystems и на таких операционных системах, как Windows NT и Windows 95/98, OS/2, AIX, HP-UX, SCO UnixWare, Linux, NUMA-Q и Sun Solaris, и сейчас поддерживает портативные устройства под управлением Windows CE и Palm OS.

DB2 Universal Database выпускается в редакциях DB2 Universal Database Enterprise - Extended Edition (платформы AIX, Solaris и Windows NT), DB2 Universal Database Enterprise Edition (платформы AIX, HP-UX, Linux, OS/2, Solaris и Windows NT), DB2 Universal Database Workgroup Edition (платформы Linux, OS/2 и Windows NT) и DB2 Universal Database Personal Edition (платформы OS/2, Linux, Windows 9x и Windows NT).

К дополнительным продуктам можно отнести:

- DB2 OLAP Server - средство для онлайновой аналитической обработки данных и реализации хранилищ данных, интегрирующее ядро Hyperion Essbase с семейством DB2 Universal Database. Работает с Hyperion Integration Server (Hyperion), Hyperion Wired for OLAP (Hyperion), Brio.Insight (Brio Technology), BUSINESSOBJECTS (Business Objects), PowerPlay (Cognos), Lotus 1-2-3 (Lotus), Excel, Internet Explorer, Visual Basic (Microsoft) и Crystal Info (Seagate);
- DB2 Connect - средство для управления соединениями различных клиентов с DB2 на AS/400;
- DB2 Universal Developer's Edition - рассчитанное на разработчиков средство для создания и тестирования приложений в архитектуре «клиент-сервер», работающих с данными DB2;
- DB2 DataJoiner - средство для доступа к реляционным и нереляционным данным, расположенным на различных платформах как к единому «образу» данных;
- DB2 Data Links Manager - средство для управления дополнительными файлами, подключаемыми к СУБД;
- DB2 Query Patroller - набор средств для создания запросов и управления ресурсами для систем принятия решений. DB2 Query Patroller получает ODBC-запросы от клиента, анализирует их и динамически распределяет запросы по различным узлам DB2 UDB Enterprise - Extended Edition;

• DB2 Net.Data - приложение, позволяющее Web-разработчикам создавать динамические Internet-приложения, используя Web Macros;

• DB2 Universal Database Satellite Edition - средство для внедрения масштабируемых мобильных решений, для управления удаленными пользователями. Поддерживает функции репликации, централизованное администрирование и средства управления через Web - DB2 Web Control Center;

DB2 Everywhere - СУБД для Palm OS и Windows CE, обеспечивающее SQL-функции и синхронизацию данных с другими реляционными базами данных, с данными Lotus Notes/Domino и PIMs.

Мы рассмотрели преимущества архитектуры «клиент-сервер», узнали, что по сравнению с информационными системами, основанными на настольных СУБД, системы, использующие серверные СУБД, обладают более высокой производительностью, меньшим сетевым трафиком, более совершенными средствами обеспечения безопасности, а также возможностями перенести на сервер баз данных часть кода, связанную с реализацией бизнес-правил и обработкой данных. Кроме того, мы обсудили возможные проблемы, которые могут возникнуть в процессе перехода от настольной СУБД к серверной, а также задачи, стоящие перед разработчиками, осуществляющими такой переход.

Мы также рассмотрели наиболее популярные на сегодняшний день серверные СУБД и признаки, которыми они характеризуются. Как правило, все или почти все данные СУБД:

• реализованы для нескольких платформ;

• обладают удобными административными утилитами;

• позволяют осуществлять резервное копирование данных;

• поддерживают несколько сценариев репликаций;

• поддерживают параллельную обработку данных в многопроцессорных системах;

• поддерживают OLAP и создание хранилищ данных;

• поддерживают выполнение распределенных запросов и транзакций;

• позволяют использовать различные средства проектирования данных для создания своих объектов;

• поддерживают средства разработки и генераторы отчетов как собственного производства, так и других производителей;

• поддерживают как минимум публикацию данных в Internet.

Существующие на сегодняшний день возможности наиболее популярных серверных СУБД отражают современные тенденции развития информационных систем, такие как использование многопроцессорных систем и распределенной обработки данных, создание распределенных систем, в том числе с использованием технологий Internet, применение средств быстрой разработки приложений, создание систем поддержки принятия решений с использованием аналитической обработки данных, а также все более повышающиеся требования к надежности и отказоустойчивости информационных систем.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

1. Дайте характеристику этапам развития средств хранения и доступа к информации.
2. Дайте обоснование многоуровневости архитектуры СУБД.
3. Приведите в соответствие уровни описания данных и уровни архитектуры СУБД.
4. Какие функциональные возможности должна иметь современная СУБД?
5. Каким образом реализуются основные функции СУБД?

6. Из каких компонент формируется типовая организация современной СУБД?
7. Укажите характерные особенности настольных СУБД.
8. Приведите примеры известных вам настольных СУБД.
9. Укажите характерные особенности серверных СУБД.
10. Какие преимущества имеет архитектура «клиент-сервер»?
11. Какие вам известны способы модернизации устаревших информационных систем?
12. Какие вам известны характерные черты современных серверных СУБД?
13. Дайте характеристику известной вам популярной серверной СУБД.

ГЛАВА 3.РЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ

Теория реляционных баз данных, как ее теперь принято называть, в настоящее время стала, безусловно, наиболее популярным методом организации системы данных при построении баз данных. СУБД (система управления базами данных), которая построена на принципах реляционной модели, называют реляционной системой управления базами данных (РСУБД). Visual FoxPro, dBase, Access, Paradox, SQL Server, Ingres, Oracle, Sybase - вот далеко не полный перечень имеющихся, в настоящее время, РСУБД.

3.1. Основные компоненты РСУБД

Реляционные базы данных состоят из таблиц, а таблицы состоят из набора столбцов (иди полей). Таблицы присутствуют как в настольных, так и клиент/серверных базах данных.

Типы данных.

В столбцах таблиц базы данных располагаются данные следующих типов:

- Strings (Строки). Используются различные типы строк, например, с однобайтовыми и двухбайтовыми символами, или строки, имеющие различную максимальную длину. Конкретные типы данных зависят от используемой СУБД.

- Numbers (Числа). Применяются различные типы числовых данных, например, целые числа, либо числа с плавающей точкой. Это связано с конкретной аппаратной платформой.

- Currency (Валюта). Специальный тип числового поля. Иногда имеет фиксированный - десятичный формат и используется для представления в клиентских приложениях "валютных" данных и/или для применения к таким данным специальных правил округления.

- Dates (Даты). Любую конкретную дату мож-но задать целым числом, например, количеством дней от Рождества Христова. Часто отсчитывается количество дней от 12/30/1899. Иногда даты хранятся в базах данных в виде целых чисел, но чаще всего для хранения дат и времени в базах данных используются числа с плавающей точкой.

- MeMos (Мемо). Этот тип используется для хранения длинных текстовых данных. В некоторых настольных СУБД длина текста ограничена определенным значением (например, 32 Кбайт).

- BLOBs (Большие двоичные объекты). Эти объекты представляют собой множество байт, содержащих любые данные (текст, графические изображения, мультимедиа, OLE-объекты, документы и т. д.). Некоторые СУБД поддерживают специальные типы больших двоичных объектов, например, поля для хранения растровых изображений, OLE-объектов, форматированного текста и т. д.

Индексы.

В большинстве реляционных баз данных реализация ключей требует создания специальных объектов базы данных, именуемых индексами.

Индекс (index) представляет собой список позиций записей, который показывает порядок их следования. Записи в реляционной таблице могут быть неупорядоченными. Тем не менее, любая запись имеет физическую позицию (положение) в файле базы данных. Эта позиция может меняться СУБД в процессе обработки данных, однако, в каждый конкретный момент времени она уникальна для каждой записи. Физическое положение (позиция) записи может измениться в процессе вставки, обновления или удаления данных, а также при выполнении некоторых манипуляций, осуществляемых сервером базы данных (таких, как компрессия файлов базы данных). Большинство современных настольных СУБД и все СУБД типа клиент/сервер поддерживают сопровождаемые индексы (maintained indexes), т. е.

индексы меняются при изменении положения записей и значений полей. В этом случае любая вставка, обновление или удаление записей приводит к тому, что производится запись самой таблицы и всех индексов, связанных с этой таблицей. Из-за этого увеличивается время, необходимое для модификации данных. Некоторые системы управления базами данных (такие как dBase III+, Fox-Base и Clipper) поддерживают индексы выражения (expression indexes). Эти индексы используются для упорядочивания данных по значению, которое получается в результате вычисления выражения на основе некоторых полей таблицы. Теперь такие индексы применяются крайне редко. Необходимо отметить, что некоторые типы полей, например memo и BLOB-поля, не могут быть проиндексированы.

Ограничения.

Большинство серверных систем управления базами данных поддерживают специальный тип объектов, которые называются ограничениями.

Ограничение (constraint) представляет собой условие, которому должны удовлетворять возможные значения, хранящиеся в поле. Например, если вы устанавливаете максимальное и минимальное значения для какого-либо поля, система управления базой данных не позволит сохранить запись, не отвечающую указанным ограничениям.

Существуют такие базы данных, которые не поддерживают ограничений (к таким СУБД, в частности, относятся dBase III, Clipper и FoxPro). При использовании подобных баз данных вам необходимо помещать код, реализующий задаваемые вами ограничения в клиентские приложения, либо применять другие объекты баз данных (например, триггеры), для того, чтобы обеспечить те же функции.

Представления.

Почти все системы управления базами данных поддерживают так называемые представления.

Представление (вид) (View) — это виртуальная таблица, обычно создаваемая как подмножество столбцов из одной и более таблиц. Представление является всего лишь описанием; само по себе оно не содержит каких-либо записей. В большинстве клиент/серверных систем управления баз данных имеются специальные визуальные средства для создания представления, которое может определять, какие именно таблицы в него включить, как их связать, какие столбцы отображать, какие ограничивающие условия (фильтры) задать для их значений.

Триггеры и хранимые процедуры.

Хранимая процедура (stored procedure) — это специальный вид процедуры, содержащейся в базе данных. Хранимые процедуры должны быть написаны на процедурном языке, специфичном для каждой СУБД. Они в ходе выполнения позволяют читать или изменять информацию в базах данных. Эти процедуры могут обращаться друг к другу, а также вызываться из клиентских приложений. Хранимые процедуры обычно используются для выполнения общих задач (например, для вычисления финансового баланса и т. д.). Они могут возвращать параметры, коды ошибок или множества строк и столбцов. Для решения некоторых специфических задач также могут применяться наборы системных процедур. Например, в Microsoft SQL Server есть набор таких хранимых системных процедур, имена которых начинаются с SP_ и XP_.

Триггеры (Triggers) также содержат код, написанный на процедурном языке. Однако они не могут вызываться непосредственно из клиентского приложения или же из хранимой процедуры. Триггеры выполняются лишь в том случае, если происходит предопределенное событие, непосредственно связанное с конкретной таблицей, например, вставка, обновление или удаление записи. С этой точки зрения, триггеры работают точно так же, как обработчики событий в Delphi, но выполняются они в адресном пространстве сервера базы данных, а не клиентского приложения. Триггер, в отличие от хранимой процедуры, связан с конкретной таблицей и запускается только тогда, когда выполняются операции вставки, удаления или

обновления в данной таблице. В большинстве систем управления базами данных может быть несколько триггеров для одного и того же события, в этом случае следует установить порядок их активизации.

Генерация первичных ключей.

Предпочтительно вставлять их с помощью клиентского приложения, что позволяет избежать дублирования возможных ключей и использовать непрерывный набор их значений. Объекты для генерации ключей отличаются в разных системах управления базами данных. Некоторые СУБД поддерживают объекты, которые хранят целое значение и правила для формирования следующего значения. Вставка записи в определенную таблицу приводит, например, к вызову триггера, который выбирает текущее значение и помещает его в ключевой столбец новой записи (после этого прежнее значение ключа в данном объекте изменяется в соответствии с заданными правилами). Такие объекты поддерживаются, в частности в Oracle (где они называются последовательностями (sequences)) и в InterBase (где они именуются генераторами (generators)). Другой способ генерации первичных ключей системой управления базами и данных заключается в том, что СУБД отводит специальные поля в таблицах, которые по определению являются первичными ключами. При вставке записи система управления базами данных заполняет это поле автоматически следующим значением. Такой подход поддерживается в Microsoft Access и в Microsoft SQL Server (где эти поля называются полями идентичности (Identity fields)), и в Corel Paradox (где их именуют полями с автоматическим приращением (Autoincrement fields)).

Пользователи и роли.

Безопасность данных, особенно предотвращение доступа к данным тех, у кого нет соответствующих прав, является серьезной проблемой. В различных системах управления базами данных она решается по-разному. Простейший способ заключается в использовании парольной защиты таблиц или отдельных столбцов в таблице (такой механизм применяется, в частности, в Corel Paradox). Более совершенный метод, поддерживаемый всеми СУБД типа клиент/сервер и некоторыми настольными системами (например, Microsoft Access), — создание списка пользователей с их именами и паролями. Любой объект базы данных принадлежит определенному пользователю, и этот пользователь может дать разрешение другому пользователю на чтение или модификацию данных, содержащихся в объекте, либо на изменение самого объекта. Вы можете просмотреть список пользователей базы данных, выбрав пункт Users (Пользователи) в SQL Server Enterprise Manager. Некоторые системы управления базами данных типа клиент/сервер также поддерживают роли. Роль (role) представляет собой набор разрешений (grants) и привилегий (priveleges). Если роль предоставляется определенному пользователю, он автоматически получает все права, предусмотренные данной ролью.

Системный каталог.

Любая реляционная база данных, в которой поддерживаются такие объекты как пользователи и роли, должна сохранять их списки. Кроме того, желательно, чтобы СУБД хранила перечень таблиц, индексов, триггеров, процедур и других объектов. Во многих случаях такая информация содержится в специальных таблицах, именуемых системными таблицами (system tables), а соответствующая часть памяти базы данных называется системным каталогом (system catalog).

В некоторых СУБД, таких как dBase и Paradox, нет системного каталога. Для того чтобы получить список таблиц такой СУБД, необходимо использовать возможности операционной системы для выбора имен файлов. Данные об индексах в этом случае могут храниться либо в индексных файлах, либо в файле, содержащем соответствующую таблицу (это зависит от версии СУБД).

Запросы.

Обработка данных и метаданных в СУБД выполняется с помощью запросов.

Запрос (query) — это требование модификации или выборки данных. Большинство современных СУБД и некоторые инструментальные средства разработки баз данных поставляются вместе с программным обеспечением для генерации запросов. Одним из популярных способов манипулирования данными является создание запросов по образцу (query by example — QBE). QBE состоит из визуального связывания таблиц и последующего выделения полей, которые должны быть получены по данному запросу. В большинстве систем управления базами данных (за исключением некоторых настольных баз данных) QBE приводит к генерации запроса на специальном языке, именуемом языком структурированных запросов (structured query language — SQL). Кроме того, можно создавать SQL-запросы непосредственно, без QBE и визуальных средств, с помощью ввода операторов SQL.

Курсоры.

Когда происходит выборка данных из базы данных, мы можем сказать, что приложение принимает некоторый набор строк, который зачастую именно так и называют row set. Очень часто этот набор строк идентичен таблице; так же как и таблица, он содержит строки и столбцы. Однако в отличие от полей и записей в базе данных, строки и столбцы в таком наборе строго упорядочены. Способ упорядочивания обычно определен запросом, на основании которого получен данный набор строк, и иногда — существующими индексами. Поскольку эти строки являются строго упорядоченными, мы можем определить текущую позицию в данном наборе строк. Указатель на текущую позицию называется курсором (cursor). Большинство современных систем управления базами данных поддерживает двунаправленные курсоры (bidirectional cursors), позволяющие перемещаться как в прямом, так и в обратном направлении в результирующем наборе строк. Однако в более старых системах управления базами данных поддерживаются лишь такие курсоры, которые позволяют перемещаться от начала к концу полученного набора строк.

SQL.

Язык структурированных запросов (structured query language - SQL) - это непроцедурный язык, который используется для манипулирования данными в реляционных СУБД. Он также является промышленным стандартом, применяемым для запросов в большинстве систем управления базами данных. Слово непроцедурный (non-procedural) означает, что с помощью SQL можно потребовать от системы управления базами данных совершения каких-либо действий, однако при этом нельзя описать алгоритм их выполнения. Любой алгоритм должен вырабатываться самой СУБД.

Расширения SQL.

Триггеры и хранимые процедуры создаются на процедурном языке, который является специфичным для каждой системы управления базами данных. В большинстве СУБД этот язык — лишь процедурное расширение SQL, т.е. представляет собой SQL плюс несколько алгоритмических операторов, таких как begin, end, if, then, else и т. д. В отличие от самого языка SQL, который отвечает требованиям стандартов ANSI, расширения SQL не стандартизованы. Любой производитель СУБД может разрабатывать собственный диалект расширенного SQL (в Oracle он называется PL/SQL, в Microsoft SQL Server — Transact-SQL, а в Microsoft Access — Jet SQL). Кроме того, некоторые расширения SQL созданы для обеспечения доступа к нереляционным и объектно-ориентированным базам данных.

Функции, определенные пользователем.

В некоторых СУБД предусмотрено использование функций, заданных пользователем (user-defined functions, UDFs). Как правило, эти функции хранятся во внешних библиотеках или на COM-серверах и должны быть зарегистрированы в базе данных. После регистрации UDF ее можно использовать в запросах, триггерах и хранимых процедурах. UDF можно написать как часть внешней библиотеки при помощи любого инструментального средства разработки, позволяющего создавать библиотеки для той платформы, на которой работает сервер базы данных. Например, можно использовать Delphi для написания UDF,

предназначенной для серверов баз данных, функционирующих в 32-разрядной системе Windows.

Транзакции.

Термин "транзакция" является основным для клиент - серверных систем управления базами данных.

Транзакция (transaction) - это группа операций с базой данных, представляющих собой некоторую логически завершенную часть обработки данных, которая может быть полностью завершена или же отменена. Существует также расширенное определение, не относящееся исключительно к базам данных: транзакция — это набор операций, которые должны быть либо выполнены все вместе, либо не выполнены совсем. Завершение транзакции (committing a transaction) означает, что все операции выполнены и зафиксированы в базе данных, а отмена (откат) транзакции (rolling back a transaction) означает, что все выполненные операции отменены, а все объекты базы данных, участвующие в этом наборе операций, возвращены к их первоначальному состоянию. Для того чтобы обеспечить возможность отмены транзакции, клиент-серверная СУБД производит запись в специальный журнал регистрации транзакций (transaction log). Это позволяет в случае отката транзакции восстанавливать данные, подвергшиеся изменению. Транзакция должна отвечать следующим требованиям, именуемым ACID - свойствами.

Элементарность (Atomicity). Означает, что транзакция должна представлять элементарную единицу обработки. Модификация данных, проведенная в ходе такой обработки, должна быть полностью выполнена (никакая ее часть не может быть не завершена).

Непротиворечивость (Consistency). Означает, что в результате транзакции все данные должны сохранять непротиворечивость (consistent state). Например, в реляционной базе данных при модификациях, осуществляемых в ходе транзакций, должны выполняться все правила, определяющие целостность на уровне ссылок (referential integrity rules).

Изолированность (Isolation). Означает, что модификации, проводимые одними транзакциями, должны быть изолированы от модификаций, проводимых одновременно другими транзакциями. Таким образом, транзакция должна "не замечать" данные, полученные в результате не связанных с нею других транзакций.

Долговечность (постоянство качеств) (Durability). Означает, что после того, как транзакция будет выполнена, результаты ее выполнения сохранятся неизменными даже в случае отказа системы.

Механизмы доступа к данным.

Во многих системах управления базами данных имеются библиотеки, содержащие специальный интерфейс прикладного программирования (application programming interface — API), который представляет собой набор функций для манипулирования данными. В СУБД для настольных систем API осуществляют лишь запись и чтение в базе данных. В СУБД типа клиент-сервер API инициирует отправку по сети запроса к серверу и получение результатов или кодов ошибок для дальнейшей их обработки клиентским приложением.

Наиболее распространенный способ доступа к данным заключается в непосредственном использовании API. Однако это означает полную зависимость вашего приложения от используемой системы управления базами данных. В этом случае переход к другой системе (например, с целью расширения памяти или для перехода от настольной системы к системе типа клиент-сервер) влечет за собой переписывание большей части программного кода клиентского приложения. Таким образом, следующим этапом в обеспечении доступа клиентского приложения к данным является создание некого универсального механизма доступа к данным, обеспечивающего для клиентского приложения стандартный набор общих функций, классов или сервисов (служб), необходимых для работы с различными системами управления базами данных. Эти

стандартные функции (классы или сервисы) должны размещаться в библиотеках, именуемых драйверами или провайдерами баз данных (data base drivers (providers)). Каждая такая библиотека реализует набор стандартных функций, классов или сервисов, используя обращения API к конкретной системе управления базами данных.

ODBC (Open Database Connectivity — открытые средства связи с базами данных) представляют собой широко применяемый пользовательский интерфейс, отвечающий требованиям стандартов ANSI и ISO, предъявляемых к интерфейсу на уровне обращений к базам данных (database call level interface (CLI)). Для доступа к реляционной базе данных посредством ODBC необходимо, чтобы был установлен так называемый администратор ODBC (ODBC Administrator) и соответствующий драйвер ODBC. Драйвер ODBC представляет собой динамически подключаемую библиотеку (dynamic-link library — DLL), которую клиентское приложение может использовать для доступа к источнику данных ODBC. В ODBC существуют драйверы для каждой поддерживаемой СУБД, так как в них используются вызовы API клиентского программного обеспечения этой СУБД. ODBC обеспечивает поддержку доступа к любым базам данных (и даже к некоторым другим файлам, например к электронным таблицам), для которых имеется соответствующий драйвер ODBC. Наряду с непосредственными вызовами API ODBC вы можете использовать также механизм доступа более высокого уровня (например, OLE DB и ADO), в котором реализованы стандартные функции, классы или службы на основе вызовов API ODBC.

OLE DB и ADO являются частью универсального доступа к данным (Universal Data Access — UDA) Microsoft. Назначением UDA является обеспечение доступа ко всем источникам данных, в том числе и нереляционным, таким как, например, файловые системы, сообщения электронной почты и многомерные базы данных. Microsoft ActiveX Data Objects (ADO) представляют собой интерфейс прикладного программирования для доступа к такого рода данным. OLE DB обеспечивает низкоуровневый интерфейс для доступа к данным. Технология ADO использует OLE DB, но интерфейс OLE DB может также применяться самостоятельно. OLE DB служит для обеспечения доступа ко всем типам данных с использованием компонентной объектной модели (Component Object Model, COM) посредством COM-интерфейсов. Для доступа к конкретным источникам данных с помощью OLE DB необходимо установить соответствующий провайдер OLE DB (OLE DB provider). Он представляет собой динамически подключаемую библиотеку (DLL), которую клиентское приложение может использовать для получения доступа к некоторому источнику данных OLE DB. Каждый провайдер OLE DB является специфическим для той или иной системы управления базами данных, так как он использует вызовы API клиентского программного обеспечения этой конкретной СУБД. Среди провайдеров OLE DB есть провайдер OLE DB корпорации Microsoft для драйверов ODBC. Этот провайдер написан на основе API ODBC, и мы можем его использовать с соответствующим драйвером ODBC. ADO содержит набор библиотек (DLLs), реализующих объектную модель ADO, которая может применяться в клиентских приложениях. Компоненты ADO Delphi — ADOExpress — созданы на основе объектов ADO, но к ним возможно прямое обращение через COM-интерфейсы и библиотеки типов.

Компоненты OLE DB. На самом верхнем уровне можно определить три главных компонента OLE DB. Это — потребители (consumers), провайдеры данных (data providers) и провайдеры сервисов (служб) (service providers). Любой компонент программного обеспечения, который использует интерфейсы OLE DB, является потребителем (consumer) OLE DB. Это может быть бизнес-приложение, инструментальное средство разработки программного обеспечения, например, Borland Delphi, сложные приложения (sophisticated applications) или же объектная модель ActiveX Data Objects, использующая интерфейсы OLE DB. Потребители используют либо те ActiveX Data Objects (ADO), которые являются интерфейсом прикладного уровня для обеспечения косвенного (indirect) доступа к данным с применением OLE DB, либо непосредственно OLE DB — для прямого доступа к данным с помощью провайдера OLE DB.

Провайдер (provider) — это часть программного обеспечения, реализующая и предоставляющая набор базовых интерфейсов OLE DB. С точки зрения OLE DB, может быть два вида провайдеров: OLE DB — провайдеры данных (data providers) и провайдеры сервисов (служб) (service providers).

Провайдер данных (data provider) представляет собой компонент программного обеспечения, "владеющий" данными. Он находится между потребителем и непосредственным массивом данных. В OLE DB все провайдеры представляют данные в табличном формате в виде виртуальных таблиц. Провайдер данных выполняет следующие задачи:

- принимает запросы, поступающие от потребителя, на доступ к данным;
- выполняет выборку или обновление данных из массива данных;
- возвращает эти данные потребителю.

Провайдер сервисов (служб) (service provider)"компонент сервисов (служб)" (service component) - реализует расширенные функциональные возможности, которые не поддерживаются обычными провайдерами данных, и сам не "владеет" данными. Этот провайдер обеспечивает сортировку, фильтрацию, управление транзакциями, обработку SQL-запросов, функции указателя (курсора) (cursor functions) и т. д. Провайдер сервисов может напрямую работать с массивами данных или же через соответствующий провайдер данных; в этом случае он выступает в роли потребителя и провайдера.

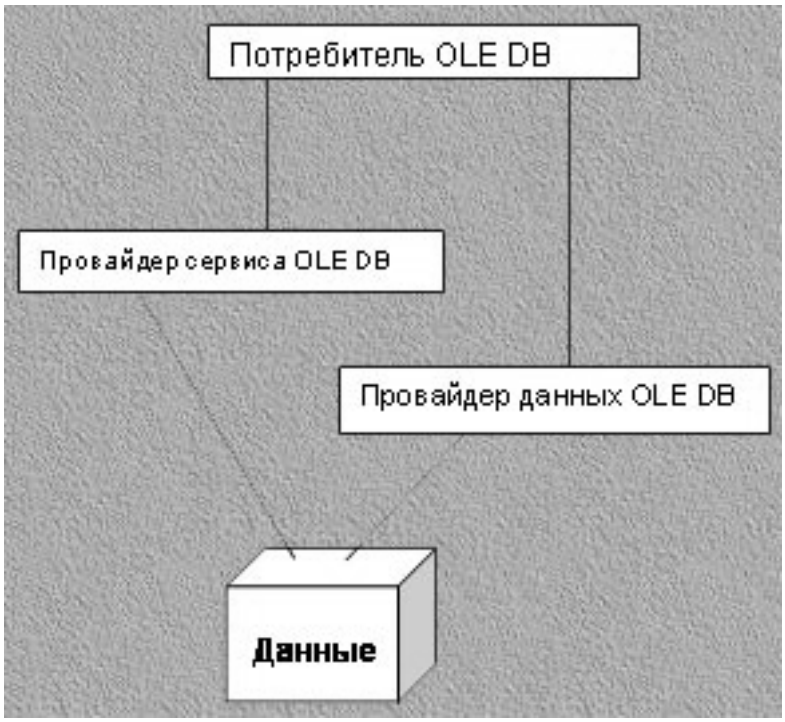

Рис.3.1. Компоненты OLE DB

OLE DB представляет собой набор интерфейсов компонентной объектной модели — Component Object Model (COM). Каждый провайдер OLE DB должен содержать объекты DataSource (Источник данных), session (Сеанс) и Rowset (Набор строк). Некоторые провайдеры OLE DB могут предоставлять несколько дополнительных объектов.

Используется потребителем данных OLE DB для связи с провайдером данных. Этот объект может быть создан различными способами, в том числе — с помощью вызова OLE-функции CoCreateInstance с идентификатором класса (class identifier, CLSID) провайдера

OLE DB, используя так называемый перечислитель (enumerator) или связывая с ним моникер файла (file moniker) либо другого источника данных. Каждый провайдер OLE DB присваивает свой собственный идентификатор класса объекту, источнику данных. Объект DataSource инкапсулирует информацию об окружении и соединении, включая имя пользователя и пароль. Основная цель объекта DataSource заключается в том, чтобы предоставить потребителю информацию из массива данных. Для создания нового сеанса (объекта session) потребители вызывают метод IDBCreateSession.CreateSession объекта DataSource.

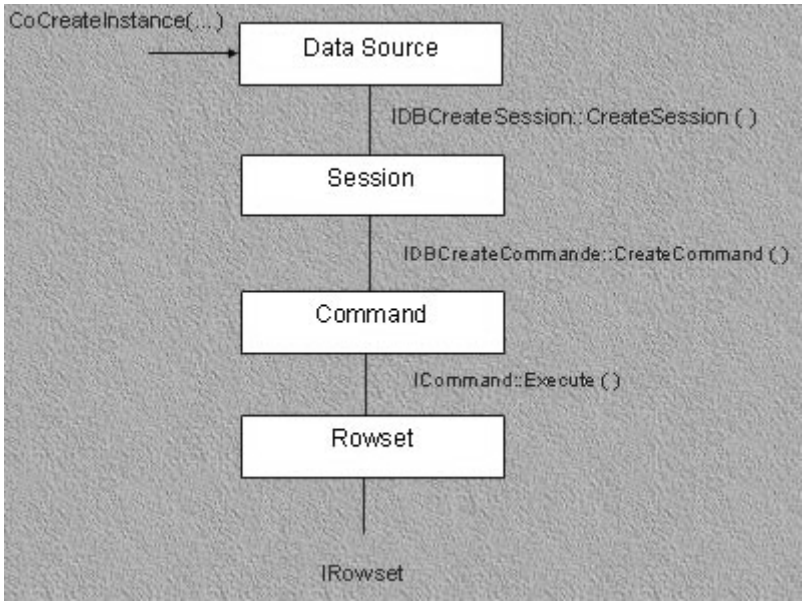

Рис.3.2. Объект DataSource

Другие механизмы доступа к данным. Среди других механизмов доступа к данным мы должны упомянуть *BDE (Borland Database Engine).* Этот механизм доступа к данным очень часто применяется в приложениях, созданных с помощью инструментальных средств разработки компании Borland (например, Delphi и C++Builder). Он является потомком библиотеки Paradox Engine, написанной для Borland Pascal и Borland C++ и предназначенной для обеспечения доступа к таблицам Paradox.

Доступ к базе данных посредством BDE обеспечивается соответствующим драйвером BDE. Для систем управления базами данных типа клиент-сервер такие драйверы носят название SQL Links. Эти драйверы содержат набор функций, именуемых BDE API, которые написаны на основе вызовов API клиентской части СУБД. В списке драйверов BDE есть драйвер, созданный на основе ODBC API, он называется ODBC Link и может использоваться лишь с соответствующим драйвером ODBC. Количество существующих драйверов BDE ограничено несколькими популярными серверами баз данных, несколькими форматами баз данных для настольных систем и сервером InterBase, поставляемым вместе с Delphi. Доступ ко всем другим базам данных осуществляется только с помощью соответствующего драйвера ODBC и ODBC Link.

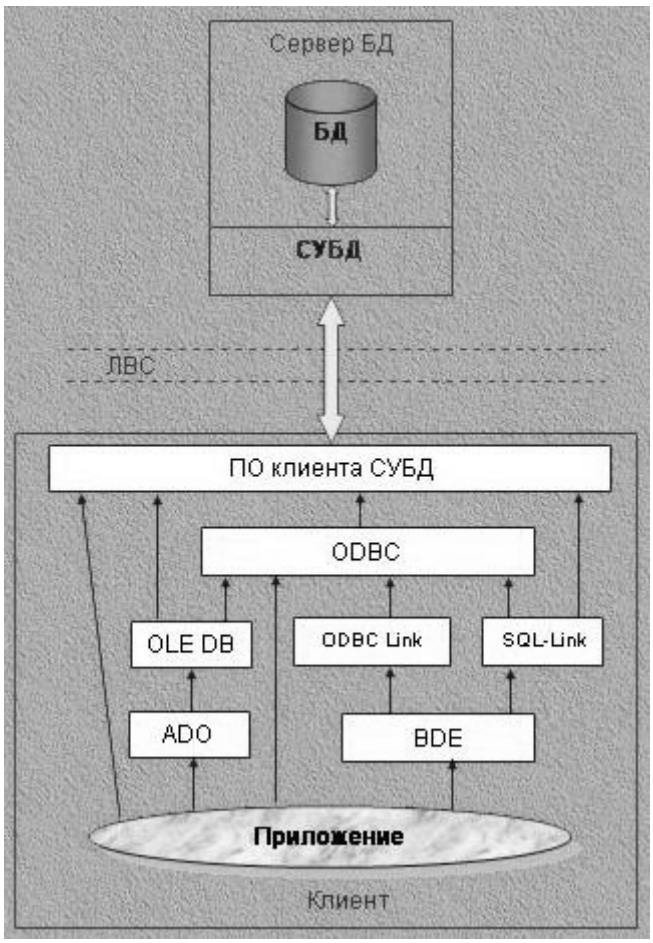

Рис.3.3. Механизмы доступа к данным

Microsoft Universal Data Access

Представляет собой стратегию обеспечения доступа ко всем типам информации, используемой в масштабах предприятия (фирмы). Она обеспечивает высокопроизводительный доступ к различным информационным источникам (включая как реляционные, так и нереляционные данные), в том числе к базам данных мэйнфреймов ISAM/VSAM, а также к иерархическим базам данных; файлам электронной почты и файловой системе; текстовым, графическим и географическим данным и так далее. Разработчики современных решений в области баз данных пытаются интегрировать данные из различных источников. Эти данные могут храниться как в различных базах данных (IBM DB2, Oracle, Microsoft SQL Server, Microsoft Access и др.), так и в других источниках данных, например, в файловых системах на различных платформах, индексно-последовательных файлах, в виде электронной почты, в электронных таблицах, инструментальном средстве управления проектом либо в виде любых других типов данных. Главная цель Microsoft Universal Data Access состоит в обеспечении доступа ко всем типам данных с помощью единой модели. Архитектура Universal Data Access включает в себя следующие элементы: ADO, OLE DB, ODBC.

3.2. Доступ к SQL-ориентированным СУБД

Каждая СУБД обеспечивает пользователя различными средствами доступа к информации, находящейся в базе данных. Доскональное знание этих средств доступа не

является обязательным для работы с SQL-ориентированными СУБД, но общее понимание принципа их работы позволит повысить эффективность работы с СУБД, а также искать и устранять тонкие ошибки, неизбежно возникающие при написании программного обеспечения, взаимодействующего с СУБД. Порядок работы с СУБД будем рассматривать на примере протокола ODBC. Помимо лучшей документированности у протокола ODBC есть еще одно несомненное преимущество – он почти «один к одному» ложится на интерфейс OCI СУБД ORACLE.

Самое первое действие при работе с реляционными СУБД – установка соединения. В момент установки СУБД проверяет права пользователя (требуется указывать имя пользователя и пароль) и настраивает программное обеспечения для работы с новым соединением. В протоколе ODBC этим действиям соответствует вызов следующих функций:

SQLAllocEnv(...)

SQLAllocConnect(...)

SQLSetConnectOption(...) // Может повториться многократно.

SQLConnect(...) или SQLDriverConnect(...)

После успешного выполнения указанной последовательности вызовов соединение программы с реляционной СУБД установлено и пользователь может начинать работу с информацией. Для выполнения тех или иных действий программа посылает СУБД различного рода запросы, написанные на языке SQL, и получает запрошенную информацию (если она требуется). Каждый запрос к СУБД проходит следующие стадии:

- разбор (parse), во время которого проверяется синтаксис запроса, подтверждаются права пользователя на выполнение этого запроса и строится т.н. план запроса. В протоколе ODBC этому этапу соответствует функция SQLAllocStmt(...) и SQLPrepare(...) (Приведенные здесь имена функций соответствуют версии 2 протокола ODBC. В версию 3 протокола внесены достаточно существенные изменения (в частности, и в имена некоторых функций), но общий принцип работы с СУБД остался прежний);
- ограничение переменных-параметров. Указанная возможность позволяет использовать ранее разобранный запрос многократно, просто подставляя в него все новые и новые значения определяющих параметров. В синтаксисе SQL для указания параметров-переменных существуют специальные конструкции (при работе через ODBC – знак вопроса, при работе через OCI - конструкция вида :<Имя> или :<Номер> и т.д.). Этот этап не является обязательным. В протоколе ODBC ему соответствует вызов (возможно, многократный) функции SQLBindParameter(...);
- собственно выполнение запроса. В протоколе ODBC ему соответствует вызов функций SQLExecute(...) или SQLExecDirect(...);
- подготовка области памяти для приема выбранной информации. Этот этап характерен исключительно для оператора SELECT. В протоколе ODBC ему соответствует вызов (возможно, многократный) функции SQLBindCol(...);
- собственно работа с выбранными данными. Физическая пересылка данных из БД в программу выполняется в протоколе ODBC с помощью функций SQLFetch(...) или SQLExtendedFetch(...). Также используется только с оператором SELECT;
- операции с выбранными данными (например, изменение полей, удаление или добавление записи). Эти операции выполняются в протоколе ODBC функцией SQLSetPos(...);
- завершение работы с выборкой – функция SQLFreeStmt(...).

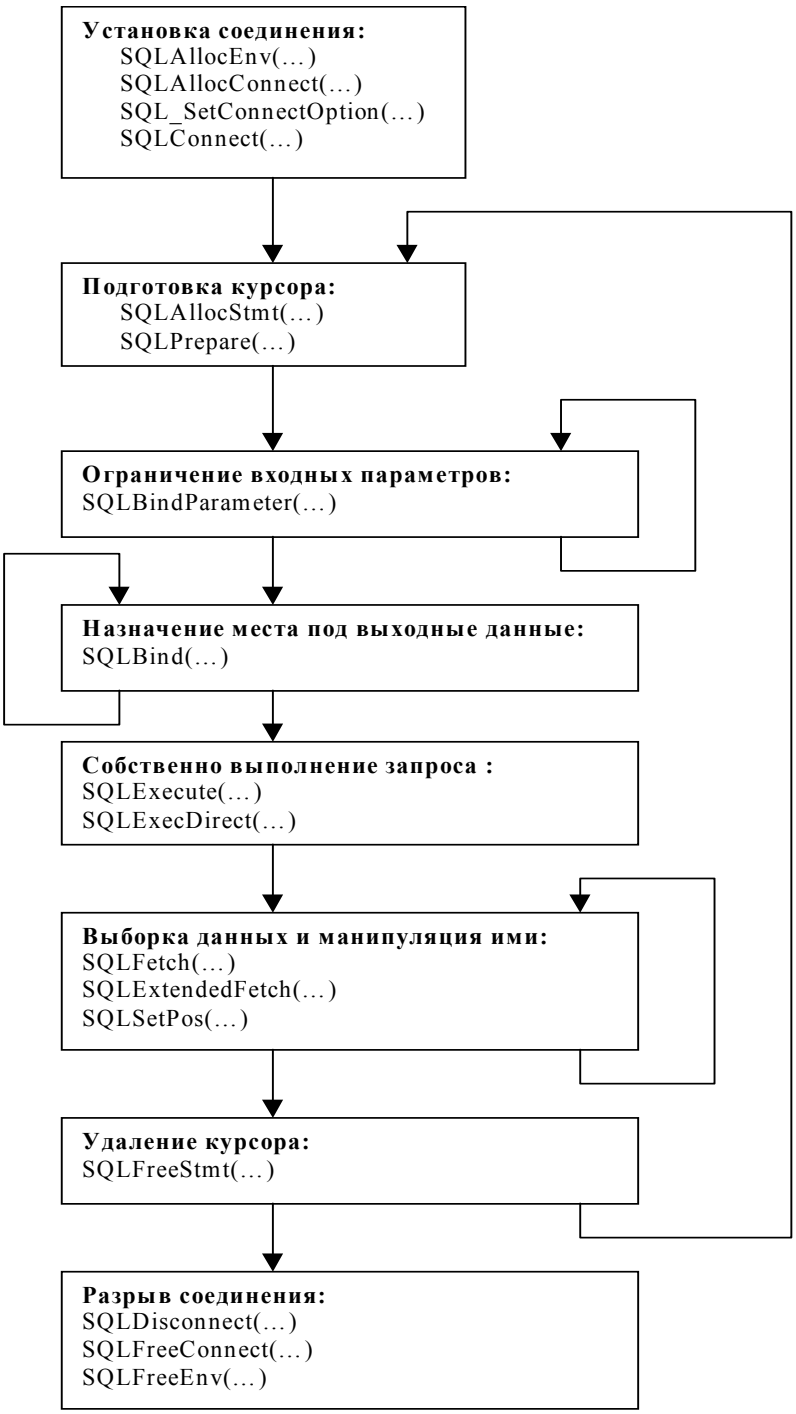

Рис.3.4. Доступ к SQL-ориентированным СУБД на примере протокола ODBC

Этот порядок многократно повторяется на каждый запрос к СУБД. При завершении работы программы выполняется закрытие соединения с СУБД:

SQLDisconnect(...)

SQLFreeConnect(...)

SQLFreeEnv(...)

При написании запросов к реляционным СУБД следует иметь в виду следующее:

Запросы на сервер базы данных передаются всегда в текстовом виде, т.е. сервер баз данных всегда работает как интерпретатор. Это позволяет формировать текст запроса непосредственно в программе.

Стадия разбора запроса (функции SQLAllocStmt(...) и SQLPrepare(...), а также SQLExecDirect(...)) являются чрезвычайно медленными операциями, поскольку они не только выполняют синтаксический анализ и компиляцию запроса, но и проверяют полномочия пользователя на его выполнение (это требует длительных поисков в словарях данных, в которых расписаны запреты и разрешения для каждого пользователя с точностью до отдельного поля в отдельной таблице), а также строит т.н. план запроса, т.е. последовательность его выполнения (это зачастую требует тонкого анализа по каждой таблице с привлечением автоматически собираемой статистики по количеству записей в таблице, частоте встречи того или иного ключа и т.д.). Поэтому при написании программ необходимо, по возможности, выполнять однократную компиляцию запроса.

Поскольку стадия разбора запроса является чрезвычайно медленной, большинство СУБД, не надеясь на грамотность разработчиков ПО, помещает все разобранные запросы в специальный системный кэш. Если новый запрос, поступивший от пользователя, уже присутствует в кэше (Факт присутствия в различных СУБД проверяется по-разному: СУБД ORACLE требует полного текстуального совпадения, SQL Server – совпадения с точностью до констант и т.д.) он повторно не разбирается. Для того, чтобы воспользоваться этой возможностью, существует общее правило – никогда без крайней необходимости не прописывать в часто выполняемых запросах никаких констант. Все вхождения констант следует заменять символом переменной-параметра, и затем ограничивать его с помощью соответствующих функций SQLBindParameter(...).

Выборка, полученная при выполнении оператора SELECT (и только его!) носит название курсора. Существует две разновидности курсоров – т.н. FORWARD_ONLY и «все остальные». Главная особенность FORWARD_ONLY-курсоров – то, что по ним возможно движение только вперед на одну запись. Остальные курсоры допускают движение во всех направлениях и на любое количество записей (SKIP +/-<Число записей>, GOTO TOP, GOTO BOTTOM, GOTO <Номер записи>). Необходимо иметь в виду следующее:

- курсоры FORWARD_ONLY работают намного быстрее. Их, по возможности, следует использовать везде, где это допустимо – в первую очередь, при печати итоговых документов.
- не все СУБД (особенно ранние версии) поддерживают не-FORWARD_ONLY-курсоры. В этих случаях протокол ODBC (или аналогичные ему средства) моделирует их «поведение» путем «выкачивания» на локальную машину выбранной информации с помощью FORWARD_ONLY - курсора. Рискнув посмотреть всего-навсего первую запись в «миллионной» таблице, можно в лучшем случае подвесить на несколько часов локальную машину, в худшем – выбить сервер. В любом случае, нельзя организовывать выборки большого размера, не убедившись предварительно, что СУБД (именно СУБД, а не протоколы типа ODBC или BDE) «умеет» работать с не-FORWARD_ONLY-курсорами.

При описании реляционной алгебры было отмечено, что нет и не может быть никакого способа узнать, в каком «месте» находится в таблице (отношении) та или иная запись (кортеж). В частности, при работе функции SQLSetPos(...) в готовую выборку можно производить добавление новых записей, при этом стандарт протокола ODBC специально подчеркивает, что нет способа узнать, в каком месте выборки эта запись окажется. В силу этого некоторые программные средства добавляют новые записи в таблицы БД весьма экзотическим образом – к примеру, записывая в таблицу запись со всеми пустыми полями, а затем организуя ее поиск и выборку. Это следует иметь в виду при трассировке обращений к БД с помощью ODBC.

3.3. Основные понятия реляционной алгебры

Как уже отмечалось, реляционная модель данных базируется на строгом математическом аппарате. Реляционная база данных, в общем случае состоит из набора таблиц (в реляционной алгебре таблица носит название отношение). Каждая таблица (отношение) состоит из записей (которые называются кортежами), а записи – из полей (которые называются атрибутами). Каждый атрибут имеет имя и значение. Значение атрибута может быть определенного типа (принадлежать определенному домену). Значение атрибута не обязано присутствовать в каждой записи (кортеже) – в некоторые кортежи этот атрибут может не входить. Факт отсутствия атрибута в кортеже обозначается специальной конструкцией NULL (Конструкция NULL обозначает не кошелек, в котором нет ни копейки денег, а отсутствие кошелька как такового).

Порядок следования записей (кортежей) и полей в таблице (отношении) никоим образом не определен и проектировщик базы данных не должен строить никаких предположений о том, в каком порядке записи будут в нем располагаться. При таком допущении две записи (два кортежа), значения атрибутов в которых полностью совпадают, не будет возможности отличить одну от другой. Для того, чтобы не возникали неоднозначности в идентификации записей (кортежей), требуется, чтобы каждый кортеж имел свой уникальный набор значений атрибутов. Как правило, не требуется, чтобы в однозначном определении использовались непременно все атрибуты записи (кортежа). Обычно выделяют среди них один или несколько атрибутов, уникальность значений которых гарантируется (например, с помощью специальных программных средств). Этот атрибут (или несколько атрибутов) носит название первичный ключ (PRIMARY KEY) и позволяет однозначно выбрать по его значению (набору значений) соответствующую запись (кортеж) в таблице (отношении). В реляционных СУБД в качестве единственного первичного ключа используется специальным образом формируемое поле (типа «счетчик», либо сгенерированное с помощью объекта SEQUENCE), обычно числовое. Это обстоятельство следует иметь в виду при проектировании реляционных СУБД или при переносе данных в реляционные СУБД из ранее разработанных систем.

Помимо собственно отношения (таблицы) в базе данных обязана присутствовать и схема отношения (описание таблицы). Схема отношения является метаданными, т.е. данными, описывающими данные. Таблиц (отношений) и схем (описаний) в базе данных может быть произвольное число (ограничения накладываются только конкретной реализацией той или иной СУБД), но каждой таблице (отношению) обязательно должна соответствовать ровно одна схема (описание) отношения и наоборот. Над таблицами (отношениями) в БД определены следующие операции:

- операция ограничения отношения;
- операция проекции отношения;
- операция прямого (или декартова) произведения отношений;
- операция соединения отношений;
- операция объединения отношений;
- операция пересечения отношений;
- операция взятия разности отношений.

Операция ограничения отношения включает в результирующую таблицу (отношение) только те записи (кортежи), которые удовлетворяют заданным условиям. В языке SQL этой операции соответствует конструкция WHERE оператора SELECT.

Операция проекции отношения включает в результирующую таблицу (отношение) только те поля (атрибуты), которые были явно заданы в операции. В языке SQL список полей (атрибутов) явно задается в списке выборки оператора SELECT.

Операция прямого произведения отношений является фундаментальной операцией реляционных СУБД. Т.е. прямое произведение таблиц (отношений) представляет собой поочередную комбинацию каждой записи из Табл1 с каждой записью Табл2. Следует иметь в виду, что таким и только таким образом соединяются в реляционной СУБД все таблицы. Если в одной таблице у Вас имеется 1000 строк, и в другой таблице имеется 1000 строк, в итоговой таблице у Вас будет 1000000 записей. Три таблицы по 1000 записей дадут 1000000000 (миллиард!) записей. Такая особенность работы – главная причина невысокой производительности реляционных СУБД на многотабличных запросах. В языке SQL операция прямого произведения задается в виде списка соединяемых таблиц в опции FROM оператора SELECT.

Операция соединения отношений является разновидностью операции прямого произведения. В отличие от прямого произведения, которое генерирует все возможные комбинации двух или нескольких таблиц, операция соединения выбирает из всех получившихся комбинаций только «осмысленные». Если, к примеру, в одной таблице у нас находятся шапки счетов-фактур, а в другой – строки счетов-фактур, то операция прямого произведения даст нам всевозможные комбинации строк и шапок, а операция соединения – только реальные счета-фактуры, в которых «свои» шапки будут соответствовать «своим» строкам. Поскольку операция соединения является разновидностью операции прямого произведения, в языке SQL она задается в виде списка соединяемых таблиц (как операция прямого произведения) и в виде специальных конструкций INNER JOIN, LEFT JOIN и RIGHT JOIN, в которых задается условие (способ) соединения таблиц. В некоторых диалектах SQL (в частности, в СУБД ORACLE) вместо этих конструкций условия соединения можно указывать непосредственно в опции WHERE оператора SELECT.

При соединении двух или нескольких таблиц (отношений) возможны ситуации, когда в обоих таблицах присутствуют одноименные поля (атрибуты). Для устранения неоднозначности в таких случаях вводится еще одна специальная операция – переименования атрибутов (конструкция AS в списке выборки оператора SELECT). Более часто, впрочем, SQL позволяет указывать т.н. квалифицированные имена, т.е. <Имя_таблицы>.<Имя_поля>, например Table1.Field1.

Операция объединения отношений представляет по сути своей слияние двух отношений (таблиц), т.е. результирующая таблица (отношение) будет содержать все записи и из первой, и из второй таблицы. Существует две разновидности операции объединения, одна из которых представлена конструкцией UNION, а другая – UNION ALL оператора SELECT. В первом случае требуется, чтобы в результирующей выборке не было одинаковых записей (кортежей) их двух разных таблиц (отношений). Вторая разновидность – это слияние двух таблиц «не глядя». Следует отметить, что не во всех диалектах SQL эти операции реализованы в полном объеме. В любом случае операция UNION (в отличие от UNION ALL) – гораздо более медленная, что необходимо учитывать при проектировании базы данных.

Операция пересечения отношений – это слияние данных из двух и более отношений (таблиц), причем в результирующее отношение попадают только те записи, которые есть в обеих таблицах (отношениях). В языке SQL эта операция представлена конструкцией INTERSECT. Эта конструкция реализована не во всех диалектах SQL и, как и UNION, достаточно медленная.

Операция взятия разности отношений заключается в выборке кортежей (записей) из первого отношения (таблицы), а затем исключении из нее всех записей, которые встречаются в другой таблице (отношении) (Следует иметь в виду, что физически никакие записи ниоткуда не удаляются, т.е. эта операция не может привести к порче базы данных). В языке SQL эта операция представлена конструкцией MINUS оператора SELECT. Эта конструкция

реализована не во всех диалектах SQL и также достаточно медленная. Как и обычная операция вычитания, операция MINUS некоммутативна, т.е. в отличие от UNION и INTERSECT результат ее работы зависит от порядка следования таблиц (отношений).

3.4. Система Microsoft SQL Server

Microsoft SQL Server - одна из наиболее мощных систем работы с базами данных в архитектуре "клиент-сервер". Особенность системы - работа сервера только в операционных системах ряда Microsoft Windows NT - NT Server 4.0, 2000 Server, Server 2003, при этом клиентская часть может взаимодействовать с сервером из Microsoft Windows 98 и других операционных систем. Рекомендуемая файловая система для SQL Server - NTFS, хотя возможна работа и в системе FAT.

3.4.1. Общая характеристика системы

В своем составе система имеет средства создания баз данных, работы с информацией баз данных, перенесения данных из других систем и в другие системы, резервного копирования и восстановления данных, развитую систему транзакций, систему репликации данных, реляционную подсистему для анализа, оптимизации и выполнения запросов клиентов, систему безопасности для управления правами доступа к объектам базы данных и пр. Система не содержит средств разработки клиентских приложений. В таблице 3.1. приведены некоторые максимальные возможности системы.

SQL Server 2000 может устанавливаться на компьютерах под управлением Windows NT и Windows 2000. Есть два способа создания таблиц, представлений, индексов и других структур базы данных. Первый способ - использовать средства графического проектирования, подобные имеющимся в Access. Второй способ состоит в написании SQL-операторов, создающих эти структуры, и передаче их на выполнение SQL Server с помощью программы SQL Query Analyzer.

SQL Server поддерживает пользовательские типы данных, которые позволяют реализовать домены. Эти типы можно использовать для определения столбцов как в средствах графического проектирования, так и в SQL-операторах. Для столбцов и пользовательских типов данных можно задавать правила. В случае пользовательского типа правило реализуется для всех столбцов, имеющих заданный тип.

Структуру таблицы можно менять с помощью графических средств или SQL-оператора ALTER TABLE. Связи можно создавать путем рисования их на диаграммах баз данных или определения внешних ключей в SQL- операторах.

Связи имеют ряд свойств, относящихся к обеспечению ссылочной целостности. Поддерживаются каскадные обновления и удаления.

SQL Server поддерживает SQL- представления. Одни представления можно использовать для обновления данных, другие - нет. Любое представление, основывающееся на одной таблице, является обновляемым, если оно не включает в себя встроенные функции или производные столбцы.

Индексы – это специальные структуры данных, используемые для повышения производительности базы данных. SQL Server автоматически создает индекс по всем первичным и внешним ключам. Дополнительные индексы создаются с помощью оператора CREATE INDEX или графического средства Manage Index. SQL Server поддерживает кластеризованные и некластеризованные индексы.

Таблица 3.1

Максимальные возможности системы

Максимальные параметры баз данных	
Наименование	Величина
Размер базы данных	1 048 516 TB
Количество объектов в базе данных	2 147 483 647
Количество экземпляров сервера на одном компьютере	16
Количество баз данных в одном экземпляре сервера	32767
Количество файлов в базе данных	32767
Количество таблиц в базе данных	ограничено количеством объектов в базе
Количество полей в таблице базы	1024
Размер файла данных	32 TB
Длина идентификаторов	128 символов
Уровень вложенных хранимых процедур	32
Уровень вложенных запросов	32
Количество некластерных индексов для одной таблицы базы	249
Количество полей в одном индексе	16
Количество байт в одном индексе	800
Количество таблиц в одном запросе	256
Количество байт в одной строке таблицы	8060

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

Хранимые процедуры могут находиться на компьютерах пользователей или в базе данных и вызываться из прикладных программ.

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

Блокировки могут налагаться с различной глубиной детализации, которая в ходе обработки может быть увеличена или уменьшена.

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

3.4.2 Типы данных системы

Для правильного проектирования баз данных необходимо знание типов данных, которые могут использоваться для полей таблиц в базе. В таблице 3.2. приведены типы данных в системе Microsoft SQL Server с разбивкой их на группы по видам.

Таблица 3.2

Типы данных в системе Microsoft SQL Server

Наименование	Описание типа данных
Двоичные данные	
binary [(n)]	максимальная длина 8 000 байт (n)
varbinary [(n)]	данные переменной длины, максимальная длина 8 000 байт (n)
Image	максимальная длина 2 147 483 647 байт
Bit	тип данных, который принимает значения 1 или 0
Символьные данные	
char [(n)]	максимальная длина 8 000 символов (n)
varchar [(n)]	тип переменной длины, максимально 8 000 символов (n)
Text	максимальная длина 1 073 741 823 символов
Символьные данные в кодировке Unicode	
nchar (n)	максимальная длина 4 000 символов (n)
nvarchar (n)	переменной длины в кодировке Unicode максимальная длина 4 000 символов (n)
Ntext	максимальная длина 1 073 741 823 символов
Числовые целые данные	
Bigint	диапазон от -922 337 203 685 4775808 до 922 337 203 685 4775807
Int	диапазон от -2 147 483 648 до 2 147 483 647
Smallint	диапазон от - 32 768 до 32 767

Окончание таблицы 3.2

Tinyint	диапазон от 0 до 255
Числовые данные с дробной частью числа	
decimal[(p[, s])]	диапазон от -1038-1 до 1038-1 с задание фиксированного количества знаков (p - всего и s -дробной части), максимальное общее количество знаков 38
Numeric	то же, что и decimal
float [(n)]	диапазон от +2.29*10-308 до +1.79*10308
Real	числа с 7-значной точностью в диапазоне от +1.18*10-38 до +3.40*1038.
Тип дата и время	
Datetime	диапазон от 1.01.1753 до 31.12.9999 с точностью 3.33 мс
Smalldatetime	диапазон от 1.01.1900 до 6.06.2079 с точностью 1 мин.
Денежный тип	
Money	диапазон от -7 203 685 477.5808 до +922 337 203 685 477.5807
Smallmoney	диапазон от -214 748.3648 до +214 748.3647
Данные специальных типов	
Timestamp	счетчик, автоматически увеличивающийся, имеющий уникальное значение для базы данных (тип binary(8) или varbinary(8))
uniqueidentifier	тип, который содержит уникальный идентификационный номер (GUID), сохраняемый как 16-битная двоичная строка
sql_variant	тип, который сохраняет значения различных типов, кроме text, ntext, timestamp и sql_variant.
Sysname	тип - синоним nvarchar , используется для ссылок на имена объектов базы данных

3.4.3. Установка системы

Установка системы Microsoft SQL Server выполняется с дистрибутивного диска запуском файла AUTORUN.EXE (который, в свою очередь, запускает программу \Sql\x86\setup\setupsql.exe). При этом начинает работать Мастер установки, который пошагово предлагает вам выбрать параметры установки системы.

1. Первый шаг - выбор компьютера для установки:
   - установить SQL Server на локальном компьютере;
   - установить на удаленном компьютере;
   - создать либо настроить виртуальный сервер.
2. Следующей шаг - выбор вида инсталляции. Возможные варианты:

- создать новую инсталляцию SQL Server;
- обновить или удалить компоненты существующей инсталляции;
- настроить виртуальный сервер;
- создать файл с информацией для автоматической установки компонентов SQL Server, которая может быть выполнена позднее.

3. Далее программа установки попросит ввести имя пользователя и название организации, а также предложит принять лицензионное соглашение.

4. После этого откроется окно Installation Definition. Оно содержит три варианта установки программного обеспечения:

- Client Tools Only - установка сетевых библиотек и средств администрирования SQL Server. Эта опция выбирается для компьютеров, которые будут использоваться для удаленного управления сервером;
- Server and Client Tools - полная установка SQL Server. Эта опция выбрана по умолчанию;
- Connectivity Only - установка сетевых библиотек и компонентов для доступа к данным (Microsoft Data Access Components, MDAC), но не средств администрирования сервера. Эта опция устанавливается для компьютеров, которые должны взаимодействовать с системой SQL Server, но не будут использоваться для администрирования SQL Server.

5. Далее следует задать установку по умолчанию или имя для именованного сервера.

6. Далее выбирается вариант установки:

- типичная,
- минимальная
- или установка пользователя (с возможностью выбора компонентов для установки) и каталог на диске компьютера для установки.

7. При выборе установки пользователя будет показано окно со списком компонентов системы и составом каждого компонента.

8. Далее необходимо задать учетные записи для запуска служб SQL Server, это может быть локальный пользователь или пользователь, зарегистрированный в домене сети и для работы с SQL Server.

9. Далее выбирается система аутентификации Windows или SQL Server.

10. Следующее окно - задание кодовой страницы и параметров сортировки данных. Здесь можно задать параметры, установленные на компьютере или отдельно заданные для системы SQL Server.

11. Следующее окно - задание используемых сервером сетевых библиотек.

12. Далее следует задать тип клиентских лицензий и их количество.

13. После этого начинается копирование файлов, и установка завершается созданием программной группы в меню Windows для работы с программами системы SQL Server.

Одно из важных новшеств системы SQL Server 2000 - возможность установки на одном компьютере нескольких экземпляров SQL Server. Экземпляр SQL Server, который устанавливается первым, называется стандартным или используемым по умолчанию; все остальные экземпляры, установленные на том же компьютере, называются именованными. Для каждого именованного экземпляра SQL Server может быть определен собственный набор баз данных и пользователей. Если на разных компьютерах установить экземпляры SQL Server с одинаковыми именами, их можно объединить в единый виртуальный сервер.

После инсталляции в группе программ Microsoft SQL Server для версии Developer Edition присутствуют пункты, показанные на рис.3.5.

Основные компоненты системы SQL Server реализуются как службы (Services) Windows. В программе SQL Server Service Manager можно управлять запуском и остановом служб, связанных с установленными компонентами системы. Ярлык этой программы появляется в области уведомлений панели задач Windows и выдает индикацию о запуске главной службы - SQL ServerAgent.

Рис. 3.5. Группа программ в меню Windows после установки системы

В состав системы SQL Server входят пять служб, для которых можно задать автоматический или ручной запуск при загрузке Windows (таблица 3.3.).

Таблица 3.3

Службы системы Microsoft SQL Server 2000

Служба	Назначение
MSSQLServer	Основное ядро SQL Server, реализует функции сервера баз данных
SQLServerAgent	Выполняет административные функции, отвечая за плановое выполнение заданий и поддержку операторов. SQL Server может работать без этой службы, но при этом ограничиваются его возможности
MS DTC (Microsoft Distributed Transaction Coordinator)	Необходима только в том случае, если в системе выполняются распределенные транзакции. Если в ней нет необходимости, можно ее не устанавливать
Microsoft Search (MS Search)	Поддерживает полнотекстовый поиск. Она генерирует каталоги и полнотекстовые индексы, а также выполняет сам поиск. Если в ней нет необходимости, можно ее не устанавливать
MSSQLServerOLAPService	Специальная служба, представляющая дополнительный компонент SQL Server - Microsoft SQL Server 2000 Analysis Services (сервер для оперативной аналитической обработки данных - OLAP)

В состав системы Microsoft SQL Server 2000 входит программа Enterprise Manager (рис.3.6.) имеющая большие возможности по администрированию и работе с базами данных. Если после запуска этой программы список серверов в ней пустой, следует зарегистрировать в ней установленные на компьютере экземпляры сервера. При использовании системы аутентификации Windows NT, запроса пароля при подключении не последует. Если используется система аутентификации SQL Server, Enterprise Manager спросит, следует ли выполнять автоматическое подключение или вы хотите, чтобы имя и пароль запрашивались у вас при каждом подключении.

После регистрации сервера для подключения к нему достаточно щелкнуть на значке "+" слева от имени сервера. Если подключение будет выполнено успешно, красный значок на пиктограмме сервера сменится на зеленый.

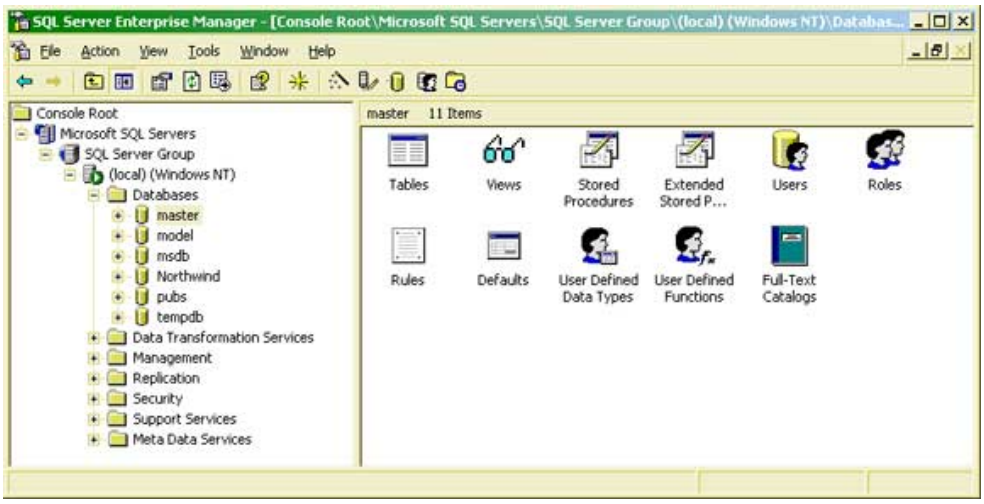

Рис. 3.6. Программа Enterprise Manager

Развернув список баз данных сервера в Enterprise Manager, мы увидим системные и установленные по умолчанию учебные базы данных. Это следующие шесть баз данных:

- master - служит для управления сервером;
- model - шаблон пользовательских баз;
- msdb - журнал выполнения заданий и расписания;
- tempdb - хранение временных таблиц и объектов;
- Northwind - пример пользовательской базы;
- Pubs - пример пользовательской базы.

Базы данных Northwind и Pubs - это учебные базы данных SQL Server, которые можно не устанавливать. Базы данных master, model, msdb, tempdb являются системными и необходимы для работы SQL Server. В программе Enterprise Manager следует просмотреть системные учетные записи, созданные в ходе установки. Для этого нужно открыть папку Logins, которая расположена в папке Security.

На сервере должны быть определены три учетные записи:

- BUILTIN\Administrators;
- ИМЯДОМЕНА\УчетнаязаписьслужбыSQLServer; (если при установке выбран Use a domain user account)
- sa.

Эти учетные записи генерируются в процессе установки SQL Server и играют очень важную роль.

Группа BUILTIN\Administrators создается исключительно при установке SQL Server в Windows NT Server или Windows NT Server Enterprise Edition, причем только при использовании системы аутентификации Windows NT. В ней представлены все члены встроенной группы Windows NT Administrators, имеющие административные разрешения на доступ к серверу.

Учетная запись sa предназначена для управления сервером. Она создается при любой установке SQL Server, поскольку без нее подключение к серверу невозможно. По умолчанию

эта запись не имеет пароля. Рекомендуется сразу же задать для нее пароль и регулярно его менять. У этой учетной записи имеются абсолютно все возможные разрешения на доступ к SQL Server и его объектам, и во всех базах данных она по умолчанию получает псевдоним dbo. При установке Desktop-версии SQL Server в Windows 9.x создается только учетная запись sa.

3.4.4. Создание базы данных

Для создания новой базы данных пользователь должен иметь права администратора или роль Database Creators. Как и многие другие операции, создание базы данных проще всего выполнить с использованием программы SQL Server Enterprise Manager.

Можно также воспользоваться программой создания базы данных, написанной на языке Transact-SQL, которую можно запустить из программы SQL Query Analyzer. Программа создания базы данных и ее таблиц может быть сгенерирована с использованием средств моделирования баз данных, например, Erwin Data Modeler (CA), Case Studio и др..

Создание базы данных в программе Enterprise Manager выполняется следующим образом.

- В окне этой программы (см. рис.3.7.) в папке Databases следует выбрать в контекстном меню или на панели инструментов команду New.
- Можно также воспользоваться мастером создания баз данных, вызываемым в пункте меню Tools окна консоли сервера.

В любом случае далее нужно задать имя базы данных, имена файлов данных и журнала транзакций и их начальный размер, величину автоматического приращения размера этих файлов.

В результате будет создана новая база по шаблону базы model. В ней будут присутствовать все группы объектов этого шаблона:

- Diagrams - схемы, отображающие связи между таблицами базы;
- Tables - папка таблиц, в которых хранится информация о таблицах базы и их индексах;

Рис. 3.7. Создание новой базы данных

- Views - папка представлений - описаний наборов данных, объединенных из нескольких таблиц в одну виртуальную таблицу;
- Stored Procedures - хранимые процедуры - список процедур на языке Transact-SQL;
- Users - сведения о владельце базы (owner) и правах пользователей, имеющих доступ к базе;

- Roles - описание типов групп пользователей;
- Defaults - описание значений по умолчанию базы и их связей с колонками таблиц;
- User Defined Data Types - описания типов данных пользователя;
- User Defined Functions - описания функций пользователя;
- Full-Text Catalog - папка для сохранения полнотекстовых индексов.

3.4.5. Создание таблиц базы данных

1. В программе Enterprise Manager в папке Table базы данных выбрать команду New.
2. В появившемся окне с названием New Table in <имя базы> on <имя SQL сервера> описать структуру таблицы, т.е. имена колонок - Column Name, тип данных в колонке - Data Type, длину данных - Length и возможность существования не заполненного информацией поля - Allow Nulls.
3. После команды «Сохранить» нужно задать имя таблицы, и она появится в списке таблиц базы.
4. Для модификации ее структуры в дальнейшем можно выбрать команду Design Table, после чего снова откроется окно описания структуры таблицы.
5. Для создания индексов в окне Design Table следует выбрать кнопку панели инструментов Manage Indexes/Keys, после чего откроется окно свойств таблицы Properties, где на третьей странице нужно описать индексы, которые могут быть уникальными или нет, кластерными (физический порядок в таблице на диске соответствует индексу) или нет. У каждой таблицы должен существовать уникальный индекс, чтобы была возможность обновления информации после модификации таблицы.

После создания всех таблиц базы, в том же окне свойств необходимо создать связи между таблицами (на второй странице окна Properties для таблиц, имеющих связи с другими таблицами), затем в папке Diagrams базы можно создать графическое представление связей между таблицами (рис.3.8.).

Для связей можно задать условия соблюдения ссылочной целостности. Эти же условия можно задать и при работе в окне Design Table.

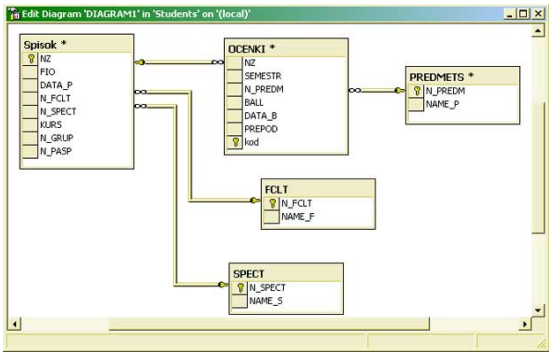

Рис.3.8. Схема базы данных

3.4.6. Работа с информацией баз данных в программе Enterprise Manager

Для добавления новых записей в таблицы, редактирования и удаления информации можно использовать команду Open table для выбранной таблицы.

При этом можно представить в окне таблицы все данные или отобрать необходимые данные с заданием условий в запросе (рис.3.9.).

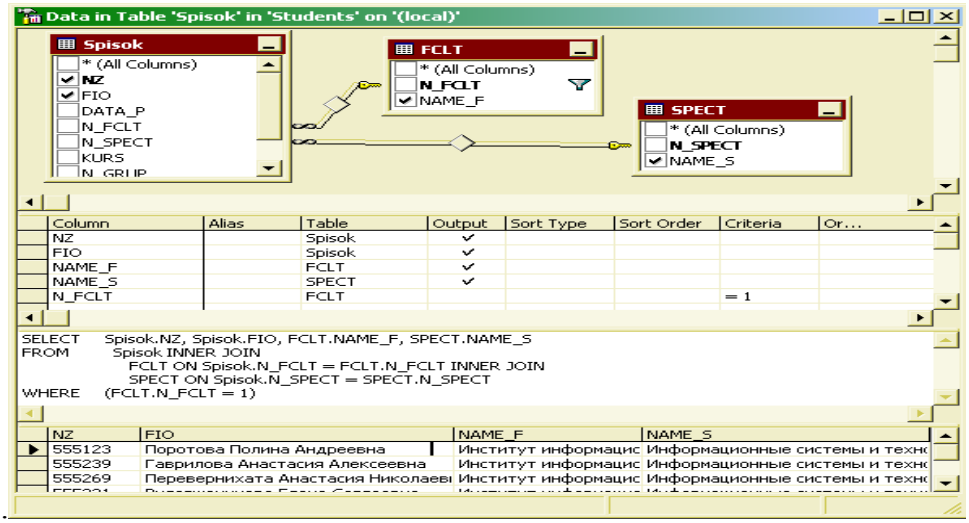

Рис. 3.9. Окно конструктора запросов

3.4.7. Разработка клиентских приложений

Основной язык работы с базой данных в системе Microsoft® SQL Server™ 2000 - Transact-SQL. Программы на этом языке генерируют такие системы, как Microsoft Visual C++®, Microsoft Visual Basic®, Microsoft Visual J++® и другие, использующие при разработки клиентских приложений программный интерфейс общего назначения (Application Programming Interface - API) ADO, OLE DB или ODBC:

ADO - Microsoft ActiveX® Data Objects поддерживает быструю разработку сложных приложений и имеет доступ к большинству компонентов системы SQL Server. По архитектуре ADO - интерфейс прикладного уровня, который использует OLE DB, библиотеку интерфейсов COM. Использование ADO ограждает прикладного разработчика от потребности программирования COM интерфейсов.

ActiveX® - это набор технологий, позволяющий компонентам программного обеспечения взаимодействовать друг с другом в сетевой среде, независимо от использовавшихся для их создания языков программирования.

OLE - связывание и внедрение объектов.

COM - технология Windows - Component Object Model.

Компоненты системы SQL Server, необходимые большинству приложений, поддерживают ADO при использовании Microsoft OLE DB Provider for SQL Server.

При разработке приложений в системе Microsoft Visual Studio.NET используется объект доступа к данным ADO.NET, предоставляющий новые возможности по работе в режиме отрыва от источника данных (соединение только на время получения и пересылки данных) [7].

OLE DB для средств, основанных на COM.

OLE DB Provider for SQL Server использует специфичные свойства провайдера, интерфейсы и методы компонентов SQL Server, не включенные в OLE DB-спецификации. Большинство этих определенных провайдером компонентов не доступно через ADO.

ODBC (Open Database Connectivity) - стандартный интерфейс, позволяющий приложениям Windows обращаться к тем источникам данных, для которых установлен

драйвер базы данных. SQL Server устанавливает свой драйвер для работы приложений с его базами.

Второй язык работы с базой данных в системе Microsoft® SQL Server™ 2000 - Xpath - язык, описанный в стандарте W3C (World Wide Web Consortium), использует XML-формат документов. Интерфейс взаимодействия с системой SQL Server - ADO API, OLE DB API. Схема взаимодействия клиентских компонентов и сервера показана на рис.3.10.

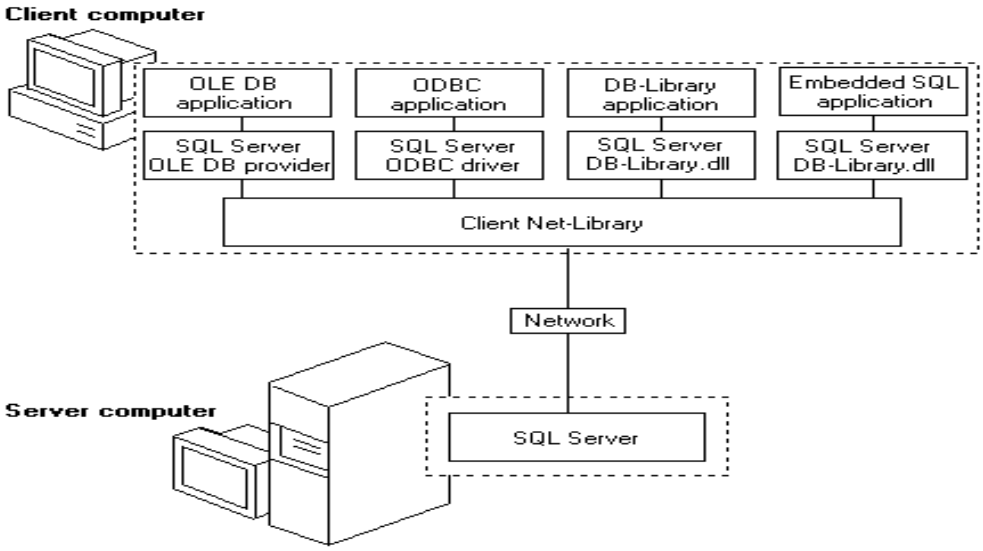

Рис. 3.10. Схема взаимодействия клиентских компонентов с сервером

ПРАКТИЧЕСКИЕ ЗАДАНИЯ

Практическое задание 3.1

Выполните генерацию информационной модели (из ERwin, практическое задание 6.2)

1. Запустите Enterprise Manager.
2. Изучите структуру СУБД MS SQL Server 2000.
3. Выберите SQL Server Group/ Нажать правую кнопку мыши/ Выбрать New SQL Server Registration (Регистрация SQL Server).
4. В качестве сервера выбрать CAPORTAL/ <ADD>

- SQL Server

Login student3

Password student3

5. Закончите регистрацию.
6. Откройте Erwin, файл с моделью данных. Перейдите на уровень Physical. Убедитесь, что все имена таблиц идентифицированы пользователем: ИмяТаблицы_№ЗачКн.
7. В качестве целевой СУБД выберите Database - SQL Server 2000.

Установите связь модели данных и SQL Server 2000 Database/Connection Database:

User student3

Password student3

Database uchet

Server caportal

8. Сгенерируйте схему базы данных uchet в Enterprise Manager:

Tools/ Schema generation

9. Проверьте правильность выполнения генерации (наличие всех таблиц в Database uchet в Enterprise Manager.

Практическое задание 3.2

- Просмотреть все таблицы базы данных uchet с помощью Query Analyzer(команда sp_help)

Порядок выполнения

1. Запустить Query Analyzer
2. Набрать в рабочем окне sp_help и нажать F5.
3. Ниже должно отобразиться окно с двумя таблицами: в первой отображаются названия объектов, их тип и владелец, а во втором – сведения о пользовательских типах

- Добавить 3-5 записей в таблицу GOT_PROD

Порядок выполнения

1. Первый способ: набрать в рабочем окне запрос

Insert into GOT_PROD(CodGotProd, NameGotProd, EdIzm) values ('115', Гвозди,кг) и нажать F5.

2. Второй способ: щелкнуть правой мышкой по названию таблицы в браузере и выбрать из контекстного меню пункт Open. Затем вручную добавить нужную запись. Не подходит для таблиц, в которых ключевое поле имеет тип счётчик.

- Вывести содержимое всех столбцов и строк таблицы CONTRAGENT. При выводе названий столбцов замените все сокращения полными словами на русском языке.

Порядок выполнения

1. Набрать в рабочем окне запрос

select contr_id as '№контрагента', contr_name as 'Название', contr_adress as 'Адрес'

from contragent

2. Нажать F5.
3. Ниже должна появиться таблица с русскоязычными заголовками

Практическое задание 3.3

Для таблицы из Практического задания 3.2 разработать триггер, запускаемый при удалении строки таблицы. Удаленная строка должна быть переписана во вспомогательную таблицу. Произвести несколько удалений строк в основной таблице и просмотреть содержимое вспомогательной таблицы.

Порядок выполнения работы

1. Изучить основные сведения о триггерах
2. Разработать собственный триггер в соответствии с вариантом задания.
3. Записать триггер в базу данных
4. Выполнить действия, приводящие к срабатыванию триггера, и убедиться в его работе
5. Получить информацию о триггере
6. Составить отчет

Практическое задание 3.4

Создайте хранимую процедуру, которая будет сохранять в базе данных информацию о новом товаре и связывать этот товар со всеми строками данного документа.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

1. Каковы базовые свойства реляционной модели данных?
2. Дайте характеристику основных компонент РСУБД.
3. Какие типы данных поддерживают реляционные СУБД?
4. Дайте характеристику основным механизмам доступа к данным реляционных СУБД.
5. Какие главные компоненты OLE DB можно определить на самом верхнем уровне?
6. Как вы понимаете принцип работы средств доступа к SQL-ориентированным СУБД?
7. Основные понятия реляционной алгебры.
8. Что из себя представляет операция ограничения отношения реляционной алгебры?
9. Что из себя представляет операция проекции отношения реляционной алгебры?
10. Что из себя представляет операция прямого (или декартова) произведения отношений реляционной алгебры?
11. Что из себя представляет операция соединения отношений реляционной алгебры?
12. Что из себя представляет операция объединения отношений реляционной алгебры?
13. Что из себя представляет операция пересечения отношений реляционной алгебры?
14. Что из себя представляет операция взятия разности отношений реляционной алгебры?
15. В чем состоят требования структурной части реляционной модели данных?
16. В чем состоят требования манипуляционной части реляционной модели данных?
17. В чем состоят требования целостной части реляционной модели данных?
18. Каковы общие свойства нормальных форм?
19. Что такое функциональная, функционально полная зависимость?
20. Каковы условия нахождения отношений в первой нормальной форме?
21. Каковы условия нахождения отношений во второй нормальной форме?
22. Каковы условия нахождения отношений в третьей нормальной форме?
23. В чем состоят общие требования обеспечения ограничений целостности?
24. Дайте общую характеристику системы Microsoft SQL Server.

ГЛАВА 4. STRUCTURED QUERY LANGUAGE - ЯЗЫК СТРУКТУРИРОВАННЫХ ЗАПРОСОВ

Structured Query Language (SQL) представляет собой непроцедурный язык, используемый для управления данными реляционных СУБД. Термин «непроцедурный» означает, что на данном языке можно сформулировать, что нужно сделать с данными, но нельзя проинструктировать, как именно это следует сделать. Иными словами, в этом языке отсутствуют алгоритмические конструкции, такие как метки, операторы цикла, условные переходы и др.

Язык SQL был создан в начале 70-х годов в результате исследовательского проекта IBM, целью которого было создание языка манипуляции реляционными данными. Первоначально он назывался SEQUEL (Structured English Query Language), затем - SEQUEL/2, а затем - просто SQL. Официальный стандарт SQL был опубликован ANSI (American National Standards Institute - Национальный институт стандартизации, США) в 1986 году (это наиболее часто используемая ныне реализация SQL). Данный стандарт был расширен в 1989 и 1992 годах, поэтому последний стандарт SQL носит название SQL92. В настоящее время ведется работа над стандартом SQL3, содержащим некоторые объектно-ориентированные расширения.

Существует три уровня соответствия стандарту ANSI - начальный, промежуточный и полный. Многие производители серверных СУБД, такие как IBM, Informix, Microsoft, Oracle и Sybase, применяют собственные реализации SQL, основанные на стандарте ANSI (отвечающие как минимум начальному уровню соответствия стандарту) и содержащие некоторые расширения, специфические для данной СУБД.

4.1. Назначение и особенности языка SQL

Язык SQL является основой многих СУБД, т.к. отвечает за физическое структурирование и запись данных на диск, а также за чтение данных с диска, позволяет принимать SQL-запросы от других компонентов СУБД и пользовательских приложений. Таким образом, SQL – мощный инструмент, который обеспечивает пользователям, программам и вычислительным системам доступ к информации, содержащейся в реляционных базах данных.

Основные достоинства языка SQL заключаются в следующем:

- стандартность – как уже было сказано, использование языка SQL в программах стандартизировано международными организациями;
- независимость от конкретных СУБД – все распространенные СУБД используют SQL, т.к. реляционную базу данных можно перенести с одной СУБД на другую с минимальными доработками;
- возможность переноса с одной вычислительной системы на другую – СУБД может быть ориентирована на различные вычислительные системы, однако приложения, созданные с помощью SQL, допускают использование, как для локальных БД, так и для крупных многопользовательских систем;
- реляционная основа языка – SQL является языком реляционных БД, поэтому он стал популярным тогда, когда получила широкое распространение реляционная модель представления данных. Табличная структура реляционной БД хорошо понятна, а потому язык SQL прост для изучения;

• возможность создания интерактивных запросов – SQL обеспечивает пользователям немедленный доступ к данным, при этом в интерактивном режиме можно получить результат запроса за очень короткое время без написания сложной программы;

• возможность программного доступа к БД – язык SQL легко использовать в приложениях, которым необходимо обращаться к базам данных. Одни и те же операторы SQL употребляются как для интерактивного, так и программного доступа, поэтому части программ, содержащие обращение к БД, можно вначале проверить в интерактивном режиме, а затем встраивать в программу;

• обеспечение различного представления данных – с помощью SQL можно представить такую структуру данных, что тот или иной пользователь будет видеть различные их представления. Кроме того, данные из разных частей БД могут быть скомбинированы и представлены в виде одной простой таблицы, а значит, представления пригодны для усиления защиты БД и ее настройки под конкретные требования отдельных пользователей;

• возможность динамического изменения и расширения структуры БД – язык SQL позволяет манипулировать структурой БД, тем самым обеспечивая гибкость с точки зрения приспособленности БД к изменяющимся требованиям предметной области;

• поддержка архитектуры клиент-сервер – SQL – одно из лучших средств для реализации приложений на платформе клиент-сервер. SQL служит связующим звеном между взаимодействующей с пользователем клиентской системой и серверной системой, управляющей БД, позволяя каждой из них сосредоточиться на выполнении своих функций.

Любой язык работы с базами данных должен предоставлять пользователю следующие возможности:

• создавать базы данных и таблицы с полным описанием их структуры;

• выполнять основные операции манипулирования данными, в частности, вставку, модификацию и удаление данных из таблиц;

• выполнять простые и сложные запросы, осуществляющие преобразование данных.

Кроме того, язык работы с базами данных должен решать все указанные выше задачи при минимальных усилиях со стороны пользователя, а структура и синтаксис его команд – достаточно просты и доступны для изучения. И, наконец, он должен быть универсальным, т.е. отвечать некоторому признанному стандарту, что позволит использовать один и тот же синтаксис и структуру команд при переходе от одной СУБД к другой. Язык SQL удовлетворяет практически всем этим требованиям.

Язык SQL является примером языка с трансформирующейся ориентацией, или же языка, предназначенного для работы с таблицами с целью преобразования входных данных к требуемому выходному виду. Он включает только команды определения и манипулирования данными и не содержит каких-либо команд управления ходом вычислений. Подобные задачи должны решаться либо с помощью языков программирования или управления заданиями, либо интерактивно, в результате действий, выполняемых самим пользователем. По причине подобной незавершенности в плане организации вычислительного процесса язык SQL может использоваться двумя способами. Первый предусматривает интерактивную работу, заключающуюся во вводе пользователем с терминала отдельных SQL-операторов. Второй состоит во внедрении SQL-операторов в программы на процедурных языках. Язык SQL относительно прост в изучении. Поскольку это не процедурный язык, в нем необходимо указывать, какая информация должна быть получена, а не как ее можно получить. Иначе говоря, SQL не требует указания методов доступа к данным. Как и большинство современных языков, он поддерживает свободный формат записи операторов. Это означает, что при вводе отдельные элементы операторов не связаны с фиксированными позициями экрана. Язык SQL может использоваться широким кругом специалистов, включая администраторов баз данных, прикладных программистов и множество других конечных пользователей.

Язык SQL – первый и пока единственный стандартный язык для работы с базами данных, который получил достаточно широкое распространение. Практически все крупнейшие разработчики СУБД в настоящее время создают свои продукты с использованием языка SQL либо с SQL-интерфейсом. В него сделаны огромные инвестиции, как со стороны разработчиков, так и со стороны пользователей. Он стал частью архитектуры приложений, является стратегическим выбором многих крупных и влиятельных организаций.

Язык SQL используется в других стандартах и даже оказывает влияние на разработку иных стандартов как инструмент определения (например, стандарт Remote Data Access, RDA). Создание языка способствовало не только выработке необходимых теоретических основ, но и подготовке успешно реализованных технических решений. Это особенно справедливо в отношении оптимизации запросов, методов распределения данных и реализации средств защиты. Начали появляться специализированные реализации языка, предназначенные для новых рынков: системы управления обработкой транзакций (OnLine Transaction Processing, OLTP) и системы оперативной аналитической обработки или системы поддержки принятия решений (OnLine Analytical Processing, OLAP). Уже известны планы дальнейших расширений стандарта, включающих поддержку распределенной обработки, объектно-ориентированного программирования, расширений пользователей и мультимедиа.

4.1.1. Применение SQL

Для успешного изучения языка SQL необходимо привести краткое описание структуры SQL-операторов и нотации, которые используются для определения формата различных конструкций языка.

Запись SQL-операторов. Оператор SQL состоит из зарезервированных слов, а также из слов, определяемых пользователем. Зарезервированные слова являются постоянной частью языка SQL и имеют фиксированное значение. Их следует записывать в точности так, как это установлено, нельзя разбивать на части для переноса с одной строки на другую. Слова, определяемые пользователем, задаются им самим (в соответствии с синтаксическими правилами) и представляют собой идентификаторы или имена различных объектов базы данных. Слова в операторе размещаются также в соответствии с установленными синтаксическими правилами.

Идентификаторы языка SQL предназначены для обозначения объектов в базе данных и являются именами таблиц, представлений, столбцов и других объектов базы данных. Символы, которые могут использоваться в создаваемых пользователем идентификаторах языка SQL, должны быть определены как набор символов. Стандарт SQL задает набор символов, который используется по умолчанию, – он включает строчные и прописные буквы латинского алфавита (A-Z, a-z), цифры (0-9) и символ подчеркивания (_). На формат идентификатора накладываются следующие ограничения:

- идентификатор может иметь длину до 128 символов;
- идентификатор должен начинаться с буквы;
- идентификатор не может содержать пробелы.

Большинство компонентов языка не чувствительны к регистру. Поскольку у языка SQL свободный формат, отдельные SQL-операторы и их последовательности будут иметь более читаемый вид при использовании отступов и выравнивания.

Язык, в терминах которого дается описание языка SQL, называется метаязыком. Синтаксические определения обычно задают с помощью специальной металингвистической символики, называемой Бэкуса-Науэра формулами (БНФ). Прописные буквы используются

для записи зарезервированных слов и должны указываться в операторах точно так, как это определено. Строчные буквы употребляются для записи слов, определяемых пользователем.

Рассмотрим, как работает SQL. Предположим, что имеется база данных, управляемая с помощью какой-либо СУБД. Для извлечения из нее данных используется запрос, сформулированный на языке SQL. СУБД обрабатывает этот запрос, извлекает запрашиваемые данные и возвращает их. Этот процесс схематически изображен на рис. 4.1.

Рис. 4.1 Обработка СУБД запроса на языке SQL

Как мы увидим позже, SQL позволяет не только извлекать данные, но и определять структуру данных, добавлять и удалять данные, ограничивать или предоставлять доступ к данным, поддерживать ссылочную целостность.

SQL сам по себе не является ни СУБД, ни отдельным продуктом. Это язык, применяемый для взаимодействия с СУБД и являющийся в определенном смысле ее неотъемлемой частью.

4.1.2. Операторы SQL

SQL содержит примерно 40 операторов для выполнения различных действий внутри СУБД. Ниже приводится краткое описание категорий этих операторов.

Data Manipulation Language (DML) DML содержит операторы, позволяющие выбирать, добавлять, удалять и модифицировать данные. Обратите внимание, что эти операторы не обязаны завершать транзакцию, внутри которой они вызваны. Операторы DML представлены в табл. 4.1.

Таблица 4.1

Операторы DML

Оператор	Описание
SELECT	Применяется для выбора данных
INSERT	Применяется для добавления строк к таблице
DELETE	Применяется для удаления строк из таблицы
UPDATE	Применяется для изменения данных

Иногда оператор SELECT относят к отдельной категории, называемой Data Query Language (DQL).

Data Definition Language (DDL) Data Definition Language содержит операторы, позволяющие создавать, изменять и уничтожать базы данных и объекты внутри них (таблицы, представления и др.). Эти операторы перечислены в табл.4.2.

Таблица 4.2

Операторы DDL

Оператор	Описание
CREATE TABLE	Применяется для добавления новой таблицы к базе данных
DROP TABLE	Применяется для удаления таблицы из базы данных
ALTER TABLE	Применяется для изменения структуры имеющейся таблицы
CREATE VIEW	Применяется для добавления нового представления к базе данных
DROP VIEW	Применяется для удаления представления из базы данных
CREATE INDEX	Применяется для создания индекса для данного поля
DROP INDEX	Применяется для удаления существующего индекса
CREATE SCHEMA	Применяется для создания новой схемы в базе данных
DROP SCHEMA	Применяется для удаления схемы из базы данных
CREATE DOMAIN	Применяется для создания нового домена
ALTER DOMAIN	Применяется для переопределения домена
DROP DOMAIN	Применяется для удаления домена из базы данных

Transaction Control Language (TCL) Операторы Transaction Control Language применяются для управления изменениями, выполненными группой операторов DML. Операторы TCL представлены в табл. 4.3.

Таблица 4.3

Операторы TCL

Оператор	Описание
COMMIT	Применяется для завершения транзакции и сохранения изменений в базе данных
ROLLBACK	Применяется для отката транзакции и отмены изменений в базе данных
SET TRANSACTION	Применяется для установки параметров доступа к данным в текущей транзакции

*Data Control Language (DCL)*Операторы Data Control Language, иногда называемые операторами Access Control Language, применяются для осуществления административных функций, присваивающих или отменяющих право (привилегию) использовать базу данных, таблицы в базе данных, а также выполнять те или иные операторы SQL. Операторы DCL представлены в табл. 4.4.

Таблица 4.4

Операторы DCL

Оператор	Описание
GRANT	Применяется для присвоения привилегии
REVOKE	Применяется для отмены привилегии

Cursor Control Language (CCL) Операторы Cursor Control Language используются для определения курсора, подготовки SQL-предложений для выполнения, а также для некоторых других операторов. Операторы CCL представлены в табл. 4.5.

Таблица 4.5

Операторы CCL

Оператор	Описание
DECLARE CURSOR	Применяется для определения курсора для запроса
EXPLAIN	Применяется для описания плана запроса. Этот оператор представляет собой расширение SQL для Microsoft SQL Server 7.0. Он не обязан выполняться в других СУБД. Например, в случае Oracle следует использовать оператор EXPLAIN PLAN
OPEN CURSOR	Применяется для открытия курсора при получении результатов запроса
FETCH	Применяется для получения строки из результатов запроса
CLOSE CURSOR	Применяется для закрытия курсора
PREPARE	Применяется для подготовки оператора SQL для выполнения
EXECUTE	Применяется для выполнения оператора SQL
DESCRIBE	Применяется для описания подготовленного запроса

Все операторы SQL имеют вид, показанный на рис. 4.2.

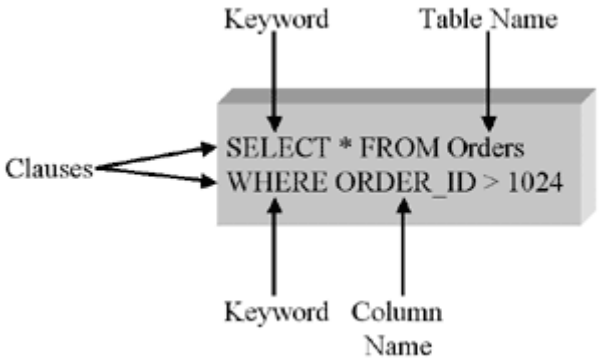

Рис.4.2. Вид оператора SQL

Каждый оператор SQL начинается с глагола, представляющего собой ключевое слово, определяющее, что именно делает этот оператор (SELECT, INSERT, DELETE...). В операторе содержатся также предложения, содержащие сведения о том, над какими данными производятся операции. Каждое предложение начинается с ключевого слова, такого как FROM, WHERE и др. Структура предложения зависит от его типа - ряд предложений содержит имена таблиц или полей, некоторые могут содержать дополнительные ключевые слова, константы или выражения.

Ключевые слова ANSI/ISO SQL92 Некоторые ключевые слова, определенные в стандарте ANSI SQL, не могут быть использованы в качестве имен объектов баз данных (таблиц, полей, имен пользователей). Эти ключевые слова приведены в приложении 1.

Стандарт SQL также включает список потенциальных ключевых слов, зарезервированных для последующих версий стандарта SQL. Эти ключевые слова приведены в приложении 2.

4.2. Выполнение SQL-операторов

Все современные серверные СУБД (а также многие популярные настольные СУБД) содержат в своем составе утилиты, позволяющие выполнить SQL-предложение и ознакомиться с его результатом. В частности, клиентская часть Oracle содержит в своем составе утилиту SQL Plus, а Microsoft SQL Server - утилиту SQL Query Analyzer. Именно этой утилитой мы воспользуемся для демонстрации возможностей SQL, а в качестве базы данных, над которой мы будем экспериментировать, возьмем базу данных NorthWind, входящую в комплект поставки Microsoft SQL Server 7.0. Можно использовать другую базу данных и любую другую утилиту, способную выполнять в этой базе данных SQL-предложения и отображать результаты (или даже написать свою, используя какое-либо средство разработки - Visual Basic, Delphi, C++Builder и др.).

4.2.1. Оператор SELECT - выбор данных

Выбор данных представляет собой наиболее часто встречающуюся операцию, выполняемую с помощью SQL. Оператор SELECT - один из самых важных операторов этого языка, применяемый для выбора данных. Синтаксис этого оператора имеет следующий вид:

SELECT column-list

FROM table-list

[WHERE where-clause]

[ORDER BY order-by-clause]

Операторы SELECT должны содержать слова SELECT и FROM; другие ключевые слова, такие как WHERE или ORDER BY, являются необязательными.

За ключевым словом SELECT следуют сведения о том, какие именно поля необходимо включить в результирующий набор данных. Звездочка (*) обозначает все поля таблицы, например:

SELECT *

Для выбора одной колонки применяется следующий синтаксис:

SELECT CompanyName

Пример выбора нескольких колонок имеет вид:

SELECT CompanyName, ContactName, ContactTitle

Если выбор данных осуществляется из нескольких таблиц и при этом выбираются одноименные поля из разных таблиц, следует ссылаться на имена таблиц для полной идентификации полей, включаемых в результирующий набор данных, например:

SELECT Customers.CompanyName, Shippers.CompanyName

Предложение FROM

Для указания имен таблиц, из которых выбираются записи, применяется ключевое слово FROM, например:

SELECT * FROM Customers

Этот запрос возвратит все поля из таблицы Customers.

Если в результирующем наборе данных нужны только поля CompanyName и ContactName, мы можем ввести следующее предложение SELECT:

SELECT CompanyName, ContactName FROM Customers

Пример запроса к более чем одной таблице приведен ниже:

SELECT Customers.CompanyName, Shippers.CompanyName

FROM Customers, Shippers

Предложение WHERE

Для фильтрации результатов, возвращаемых оператором SELECT, можно использовать предложение WHERE, синтаксис которого имеет вид:

WHERE expression1 [{AND | OR} expression2 [:]]

Например, вместо получения полного списка продуктов можно ограничиться только теми из них, у которых значение поля CategoryID равно 4:

SELECT * FROM Products

WHERE CategoryID = 4

В предложении WHERE можно использовать различные выражения, например:

SELECT * FROM Products

WHERE CategoryID = 2 AND SupplierID > 10

или:

SELECT ProductName, UnitPrice FROM Products

WHERE CategoryID = 3 OR UnitPrice < 50

или:

SELECT ProductName, UnitPrice FROM Products

WHERE Discontinued IS NOT NULL

Выражение 'IS NOT NULL' означает, что соответствующая колонка результирующего набора данных не может содержать пустых значений.

В предложении WHERE можно использовать один из шести операторов отношений, определенных в SQL. Эти операторы приведены в табл. 4.6.

Таблица 4.6

Операторы отношений, определенные в SQL

Оператор	Описание
<	Меньше
<=	Меньше или равно
<>	Не равно
=	Равно
>	Больше
>=	Больше или равно

Помимо перечисленных выше простых операторов сравнения, можно использовать и специальные операторы сравнения, приведенные в табл. 4.7.

Таблица 4.7

Специальные операторы сравнения

Оператор	Описание
ALL	Применяется совместно с операторами сравнения при сравнении со списком значений
ANY	Применяется совместно с операторами сравнения при сравнении со списком значений
BETWEEN	Применяется при проверке нахождения значения внутри заданного интервала (включая его границы)
IN	Применяется для проверки наличия значения в списке
LIKE	Применяется при проверке соответствия значения заданной маске

Приведем несколько примеров применения этих операторов. Для сопоставления данных с маской применяется ключевое слово LIKE:

```
SELECT CompanyName, ContactName
 FROM Customers
WHERE CompanyName LIKE 'M%'
```

В данной маске символ '%' (процент) заменяет любую последовательность символов, а символ '_' (подчеркивание) - один любой символ. Тот же самый результат может быть получен следующим способом:

```
SELECT CompanyName, ContactName
FROM Customers
WHERE CompanyName BETWEEN 'M' AND 'N'
```

В последнем примере мы можем расширить область поиска. В частности, при поиске компаний с именами, начинающимися с букв от A до C, можно выполнить следующий оператор SELECT:

```
SELECT CompanyName, ContactName
 FROM Customers
 WHERE CompanyName BETWEEN 'A' AND 'D'
```

Используя оператор IN, можно задать список значений, в котором должно содержаться значение поля:

```
SELECT CompanyName, ContactName
FROM Customers
WHERE CustomerID IN ('ALFKI', 'BERGS', 'VINET')
```

Логические операторы AND, OR и NOT

Мы уже рассматривали пример применения оператора AND для логических операций, связанных с требованием, чтобы запись удовлетворяла двум разным критериям. Рассмотрим следующий запрос:

```
SELECT CompanyName, ContactName
 FROM Customers
WHERE CompanyName LIKE 'S%' AND Country = 'USA'
```

Результатом выполнения этого запроса будет список заказчиков, находящихся в США, название которых начинается с буквы S.

Оператор OR позволяет выбрать записи, удовлетворяющие хотя бы одному из перечисленных условий, в то время как оператор NOT используется для исключения из

набора данных записей, удовлетворяющих данному условию. Например, можно применить оператор OR для поиска всех заказчиков, либо находящихся в Калифорнии, либо имеющих название, начинающееся с буквы S (и при этом находящихся где угодно):

```
SELECT CompanyName, ContactName
FROM Customers
 WHERE CompanyName LIKE 'S%' OR Region='CA'
```

В этом случае результирующий набор данных будет содержать записи, в которых значение поля CompanyName удовлетворяет первому условию, плюс все записи, в которых значение поля Region удовлетворяет второму условию.

Теперь рассмотрим пример применения оператора NOT. Для исключения некоторых заказчиков из результирующего набора данных можно использовать запрос вида:

```
SELECT CompanyName, ContactName
 FROM Customers
 WHERE Country NOT IN ('USA', 'UK')
```

В результате выполнения этого запроса мы получим список заказчиков из всех стран, кроме США и Великобритании.

Предложение ORDER BY

Предложение ORDER BY (необязательное) применяется для сортировки результирующего набора данных по одной или нескольким колонкам. Для определения порядка сортировки используются ключевые слова ASC (по возрастанию) или DESC (по убыванию). По умолчанию данные сортируются по возрастанию. Синтаксис предложения ORDER BY имеет вид:

```
ORDER BY column1 [{ASC | DESC}]
[, column2 [{ASC | DESC}] [,:]
```

Например, для сортировки сотрудников по фамилии и затем по имени следует использовать следующий SQL-запрос:

```
SELECT LastName, FirstName, Title
FROM Employees
ORDER BY LastName, FirstName
```

Если сортировка данных требуется в убывающем порядке (например, требуется список продуктов в порядке убывания цен), используется ключевое слово DESC:

```
SELECT ProductName, UnitPrice
 FROM Products
ORDER BY UnitPrice DESC
```

Связывание таблиц

Как мы уже убедились, можно создавать запросы, позволяющие извлечь данные из нескольких таблиц. Одна из возможностей сделать это заключается в связывании таблиц по одному или нескольким полям. Обратите внимание на то, что без связывания таблиц в результате запроса получится набор данных, содержащий все возможные комбинации строк каждой из исходных таблиц (известное также как декартово произведение):

```
SELECT ProductName, CategoryName
FROM Products, Categories
```

в то время как запрос, показанный ниже, приводит к отображению списка продуктов с указанием, к какой категории принадлежит данный продукт:

```
SELECT ProductName, CategoryName
FROM Products, Categories
```

WHERE Products.CategoryID = Categories.CategoryID

Можно сравнить результаты этих двух запросов.

В общем случае синтаксис для связывания таблиц имеет вид:

SELECT column-list

FROM table1, table2

WHERE table1.column1=table2.column2

Следующие несколько примеров связывания таблиц характерны для Microsoft Access и Microsoft SQL Server и могут не работать с другими СУБД, однако мы полагаем, что иллюстрируемая ими функциональность достаточно важна.

Существует несколько типов связывания таблиц. Например, следующий оператор SQL осуществляет так называемое внутреннее соединение таблиц (inner join) - в этом случае в результирующем наборе данных содержатся записи, в которых значения в связанных полях совпадают:

SELECT ProductName, CategoryName

FROM Products INNER JOIN Categories

ON Products.CategoryID = Categories.CategoryID

Так называемые внешние соединения (outer joins) позволяют нам включить в результат запроса все строки из одной таблицы и соответствующие им строки из другой таблицы. Например:

SELECT ProductName, CategoryName

FROM Products LEFT OUTER JOIN Categories

ON Products.CategoryID = Categories.CategoryID

Это было так называемое левое внешнее соединение (left outer join). Существуют также правые внешние соединения (right outer join), возвращающие все строки из второй (то есть правой) таблицы и соответствующие им строки из другой таблицы:

SELECT ProductName, CategoryName

FROM Products RIGHT OUTER JOIN Categories

ON Products.CategoryID = Categories.CategoryID

Комбинируя левое и правое внешние соединения, можно получить полное внешнее соединение, возвращающее все данные из обеих таблиц:

SELECT ProductName, CategoryName

FROM Products FULL OUTER JOIN Categories

ON Products.CategoryID = Categories.CategoryID

Для получения всех комбинаций строк из обеих таблиц (декартова произведения) можно использовать ключевое слово CROSS JOIN без указания связываемых полей:

SELECT ProductName, CategoryName

FROM Products CROSS JOIN Categories

Если в запросе участвуют более трех таблиц, можно использовать вложенные соединения.

Предложение GROUP BY

Для вычисления суммарных значений на основе данных одной или нескольких таблиц можно использовать предложение GROUP BY, имеющее такой синтаксис:

GROUP BY {column1} [, :]

Например, следующий запрос связывает две таблицы, сортирует их по полю CustomerID, для каждого значения CustomerID создает одну строку в результирующем

наборе данных и вычисляет количество значений поля OrderID для каждого значения CustomerID:

```
SELECT Customers.CustomerID,
COUNT (Orders.OrderID)
FROM Customers INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
GROUP BY Customers.CustomerID;
```

В приведенном выше примере запроса мы использовали в предложении SELECT агрегатную функцию COUNT, вычисляющую количество значений. В табл. 4.8 указан список наиболее часто используемых агрегатных функций.

Таблица 4.8

Наиболее часто используеме агрегатные функции

Функция	Назначение
AVG	Вычисляет среднее
COUNT	Вычисляет количество непустых значений в данной колонке
MAX	Вычисляет наибольшее значение в колонке
MIN	Вычисляет наименьшее значение в колонке
SUM	Вычисляет сумму значений в колонке
STDEV	Вычисляет несмещенное стандартное отклонение для данной колонки. Эта функция используется в Microsoft Access и Microsoft SQL Server. В случае Oracle она называется STD
STDEVP	Вычисляет смещенное стандартное отклонение для данной колонки. Эта функция используется в Microsoft Access и Microsoft SQL Server
VAR	Вычисляет несмещенную дисперсию отклонения для данной колонки. Эта функция используется в Microsoft Access и Microsoft SQL Server. В случае Oracle она называется VARIANCE
VARP	Вычисляет смещенную дисперсию отклонения для данной колонки. Эта функция используется в Microsoft Access и Microsoft SQL Server

Помимо перечисленных выше агрегатных функций можно использовать также математические и строковые функции, приведенные в приложении 3.

Предложение HAVING

Предложение HAVING имеет назначение, сходное с предложением WHERE, но используется с агрегатными данными. Например:

```
SELECT Customers.CustomerID,
COUNT (Orders.OrderID)
FROM Customers INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
GROUP BY Customers.CustomerID
HAVING COUNT(Orders.OrderID) >= 10
```

Этот запрос аналогичен предыдущему, но в результирующий набор данных включены только заказчики, разместившие десять или более заказов.

Ключевые слова ALL и DISTINCT

До этого момента мы рассматривали, как извлечь все или заданные колонки из одной или нескольких таблиц. Для управления выводом дублирующихся строк результирующего набора данных можно использовать ключевые слова ALL или DISTINCT в предложении SELECT. Ключевое слово DISTINCT указывает, что строки результирующего набора данных должны быть уникальны, тогда как ключевое слово ALL указывает, что возвращать следует все строки. Например, для извлечения названий стран, в которых имеются заказчики, можно использовать следующий запрос:

```
SELECT DISTINCT Country FROM Customers
```

Отметим, что ключевое слово ALL используется по определению. Если в запросе требуется вывести более одной колонки, и при этом использовано слово DISTINCT, то результирующий набор данных будет содержать различные строки, но некоторые значения одного и того же поля в разных строках могут совпадать.

4.2.2. Модификация данных

До сих пор мы изучали операторы SQL для извлечения данных. Помимо этого язык SQL может быть использован для обновления и удаления данных, копирования записей в другие таблицы и выполнения многих других операций. Ниже мы рассмотрим операторы UPDATE, DELETE и INSERT, используемые для решения некоторых из этих задач.

Оператор UPDATE

Для изменения значений в одной или нескольких колонках таблицы применяется оператор UPDATE. Синтаксис этого оператора имеет вид:

```
UPDATE tableSET column1 = expression1 [, column2 = expression2] [,:]
[WHERE criteria]
```

Выражение в предложении SET может быть константой или результатом вычислений. Например, для повышения цен всех продуктов, стоящих меньше 10 долл., можно выполнить следующий запрос:

```
UPDATE Products
SET UnitPrice = UnitPrice * 1.1
WHERE UnitPrice < 10
```

Оператор DELETE

Для удаления строк из таблиц следует использовать оператор DELETE, синтаксис которого имеет вид:

```
DELETE
```

```
FROM table
[WHERE criteria]
```

Внимание! Предложение WHERE не является обязательным, но если вы забудете его включить, из таблицы будут удалены все записи.

Например, для удаления из списка всех продуктов, которые больше не поставляются, можно выполнить следующий запрос:

```
DELETE
FROM Products
WHERE Discontinued = 1
```

Полезно использовать оператор SELECT с тем же синтаксисом, что и оператор DELETE, чтобы проверить, какие именно записи будут удалены, прежде чем действительно их удалять. Ниже показан оператор SELECT для приведенного выше запроса на удаление данных:

```
SELECT ProductName
FROM Products
WHERE Discontinued = 1
```

Можно использовать в предложении WHERE более сложный критерий для определения того, какие записи должны быть удалены. Предположим, нам нужно удалить из списка клиентов тех из них, кто не имел заказов до определенной даты. Сначала для этого следует выполнить следующий SELECT, чтобы определить, что именно мы удаляем:

```
SELECT CompanyName
FROM Customers
WHERE Customers.CustomerID NOT IN
(SELECT CustomerID FROM Orders WHERE OrderDate > 01/01/96)
```

а затем заменить оператор SELECT на оператор DELETE:

```
DELETE FROM Customers
WHERE Customers.CustomerID NOT IN
(SELECT CustomerID FROM Orders WHERE OrderDate > 01/01/96)
```

Замечание. При использовании в операторах SQL даты или времени, а также полей, содержащих такие данные, следует уточнить синтаксис таких предложений в документации из комплекта поставки используемой СУБД.

Оператор INSERT

Для добавления записей в таблицы следует использовать оператор INSERT, синтаксис которого имеет вид:

```
INSERT [INTO] table
( [column_list]
{ VALUES ( { DEFAULT | NULL | expression }
 } [, :] )
```

Например, для добавления нового клиента в таблицу Customers можно использовать следующий запрос:

```
INSERT INTO Customers
(CustomerID, CompanyName)
VALUES
('XYZFO', 'XYZ Deli')
```

4.2.3. Модификация метаданных

Существует несколько операторов SQL для управления метаданными, используемых для создания, изменения или удаления баз данных и содержащихся в них объектов (таблиц, представлений и др.). Мы рассмотрим некоторые из них: CREATE TABLE, ALTER TABLE и DROP.

Оператор CREATE TABLE

Для создания новой таблицы необходимо использовать оператор CREATE TABLE, синтаксис которого имеет вид:

```
CREATE TABLE table
( column1 type1 [(size1)][CONSTRAINT _
column-constraint1]
[, column2 type2 [(size2)][CONSTRAINT _
column-constraint2]
[, ...]]
[CONSTRAINT table-constraint1 _
[,table-constraint2 [, ...]]]);
```

В этом операторе следует указать имя поля, тип данных для него (тип данных должен поддерживаться данной СУБД), длину (для некоторых типов полей) и, если нужно, серверные ограничения (с применением ключевого слова CONSTRAINT). Например, следующий запрос создает таблицу с именем Simple с четырьмя колонками - LastName, FirstName, EMail и HomePage:

```
CREATE TABLE Simple
(FirstName varchar(50) NOT NULL,
LastName varchar(50) NOT NULL,
EMail varchar(50),
HomePage varchar(255)
 )
```

Мы можем расширить эту таблицу добавлением поля PersonID, которое будет использовано как первичный ключ:

```
CREATE TABLE Simple
( PersonID Integer NOT NULL PRIMARY KEY,
 FirstName varchar(50) NOT NULL,
LastName varchar(50) NOT NULL,
 EMail varchar(50),
HomePage varchar(255)
 )
```

и указать, что комбинация полей LastName и FirstName должна быть уникальна:

```
CREATE TABLE Simple
( PersonID Integer NOT NULL PRIMARY KEY,
FirstName varchar(50) NOT NULL,
LastName varchar(50) NOT NULL,
EMail varchar(50),
HomePage varchar(255),
CONSTRAINT SimpleConstraint UNIQUE
```

(FirstName, LastName)

)

Используя предложение SELECT и ключевое слово INTO, мы можем создавать новые таблицы, основанные на условии, указанном в предложении WHERE. Например:

```
SELECT *
INTO NewOrders
FROM Orders
WHERE OrderDate > 1/1/97
```

Этот запрос создаст новую таблицу NewOrders и заполнит ее данными о заказах начиная с 1 января 1997 года.

Оператор ALTER TABLE

Для изменения структуры существующей таблицы можно использовать оператор ALTER TABLE. Применяя его, можно добавить или удалить поле или серверное ограничение. Существует четыре разновидности оператора ALTER TABLE.

Первая разновидность этого оператора используется для добавления колонки к таблице, и ее синтаксис имеет вид:

```
ALTER TABLE table ADD [COLUMN] column datatype
[(size)]
[CONSTRAINT sinlge-column-constraint]
```

В запросах такого вида определяется имя таблицы, имя нового поля, его тип данных и, если нужно, размер. Помимо этого можно указать серверное ограничение, связанное с данным полем. Например, для добавления поля Phone к таблице Simple, созданной ранее, можно выполнить следующий запрос:

```
ALTER TABLE Simple ADD Phone varchar(30)
```

Вторая разновидность оператора ALTER TABLE применяется для добавления серверных ограничений к таблице, а ее синтаксис имеет вид:

```
ALTER TABLE table ADD CONSTRAINT constraint
```

Такие запросы позволяют только добавлять индексы, позволяющие использовать соответствующие поля в качестве первичных или внешних ключей.

Третья разновидность предложения ALTER TABLE применяется для удаления поля из таблицы:

```
ALTER TABLE table DROP [COLUMN] column
```

Ключевое слово COLUMN использовать не обязательно. Например:

```
ALTER TABLE Simple DROP Phone
```

Обратите внимание на то, что для удаления проиндексированных полей следует сначала удалить индекс. Это можно сделать с помощью четвертой разновидности предложения ALTER TABLE:

```
ALTER TABLE table DROP CONSTRAINT index
```

Ниже приведен пример такого запроса:

```
ALTER TABLE Simple DROP CONSTRAINT PrimaryKey
```

Оператор DROP

Для удаления таблиц или индексов можно использовать оператор DROP, имеющий две разновидности. Первая из них применяется для удаления таблицы из базы данных:

```
DROP TABLE table
```

Вторая разновидность используется для удаления индекса:

```
DROP INDEX index ON table
```

Другие операторы SQL

Как было отмечено ранее, существует около 40 операторов SQL. Мы рассмотрели большинство из них. Некоторые из не рассмотренных нами операторов перечислены ниже:

- операторы CREATE, такие как CREATE DATABASE, CREATE VIEW и CREATE TRIGGER;
- операторы ALTER, такие как ALTER DATABASE, ALTER VIEW и ALTER TRIGGER;
- операторы DROP, такие как DROP DATABASE, DROP VIEW и DROP TRIGGER;
- BEGIN TRANSACTION, COMMIT TRANSACTION и ROLLBACK TRANSACTION для выполнения группы нескольких операторов как единой логической группы;
- DECLARE CURSOR, OPEN и FETCH для работы с курсорами;
- GRAND и REVOKE для добавления или удаления прав на использование объектов базы данных,
- а также CREATE USER, ALTER USER, DROP USER, CREATE GROUP, ALTER GROUP и DROP GROUP для управления списком пользователей и групп пользователей.

4.3. Создание и использование объектов баз данных средствами SQL

Триггеры и хранимые процедуры обычно пишутся на языках программирования, представляющих собой процедурные расширения языка SQL. Эти расширения содержат операторы, позволяющие описывать алгоритмы, например do:while, if:then:else, отсутствующие в самом языке SQL (SQL - непроцедурный язык, и на нем можно сформулировать задание, но нельзя описывать алгоритмы его выполнения). В отличие от языка SQL, подчиняющегося стандарту, его процедурные расширения никак не стандартизованы, и разные СУБД используют разные синтаксические конструкции для реализации одних и тех же алгоритмических конструкций.

Для иллюстрации того, как можно использовать представления, триггеры и хранимые процедуры, мы выбрали Microsoft SQL Server 2000 и базу данных NorthWind, входящую в комплект поставки этой СУБД.

4.3.1. Представления

Представление - это виртуальная таблица, обычно содержащая набор колонок одной или нескольких таблиц. В действительности представление содержит не данные, а лишь SQL-запрос типа SELECT, указывающий, какие именно данные и из каких таблиц нужно взять при обращении к этому представлению. С этой точки зрения представление - это хранимый запрос.

В большинстве случаев представления используются для обеспечения безопасности данных. Например, некоторые категории пользователей могут иметь доступ к представлению, но не к таблицам, данные которых его формируют; кроме того, SQL-запрос может содержать параметр USER (имя, под которым зарегистрировался пользователь), и в этом случае данные, доступные при обращении к представлению, будут зависеть от имени пользователя.

Ниже перечислены основные характеристики представлений:

- представления ведут себя подобно таблицам;
- представления не содержат данных;
- представления могут использовать данные более чем из одной таблицы.

Для создания представления мы можем использовать SQL-предложение CREATE VIEW, для его модификации - предложение ALTER VIEW, а для удаления его - предложение DROP VIEW.

Предложение CREATE VIEW.

Синтаксис предложения для создания представления напоминает SQL-предложение SELECT с несколькими дополнительными ключевыми словами. Ниже приведен его упрощенный синтаксис:

CREATE VIEW view_name

[WITH ENCRYPTION]

AS

select_statement

Аргумент view_name указывает на имя представления. Ключевое слово [WITH ENCRYPTION], используемое в Microsoft SQL Server, позволяет скрыть исходный текст предложения CREATE VIEW в таблице syscomments.

Ключевое слово AS указывает, какой запрос SELECT реально будет выполняться при обращении к представлению. Этот запрос не может содержать ключевых слов ORDER BY, COMPUTE или COMPUTE BY, INTO и не может ссылаться на временную таблицу.

Предложение ALTER VIEW.

Предложение ALTER VIEW имеет тот же синтаксис, что и предложение CREATE VIEW. Его основное назначение - внести изменения в уже имеющееся представление.

Предложение DROP VIEW.

Это предложение используется для удаления представления из базы данных. При удалении таблицы из базы данных удаляются и все представления, ссылающиеся на нее. Используя это предложение, мы должны указать имя удаляемого представления. После того как представление удалено, вся информация о нем удаляется из системных таблиц.

Еще один случай, когда представление нужно удалить, может возникнуть при условии, что структура таблиц, на которых оно основано, изменилась после создания представления. В этом случае можно удалить представление и затем создать его заново с помощью предложения CREATE VIEW.

Предложение CREATE VIEW используется для создания представлений, позволяющих извлекать данные, удовлетворяющие определенным требованиям. Представление создается в текущей базе данных и хранится как ее отдельный объект.

Создание и использование представлений.

Наилучший способ создать представление - создать запрос SELECT и, проверив его, добавить недостающую часть предложения CREATE VIEW. Давайте рассмотрим исходный текст представления Products by Category в базе данных NorthWind (листинг 4.1).

Первая строка, выделенная жирным шрифтом, представляет собой то, чем отличается SQL-предложение для создания представления от обычного запроса SELECT, выполняющего работу по выбору данных. Предложение SELECT, содержащееся в этом представлении, выбирает поля из двух таблиц - поле CategoryName из таблицы CATEGORIES и поля ProductName, QuantityPerUnit, UnitsInStock, Discontinued из таблицы PRODUCTS. После этого данные двух таблиц связываются по полю CategoryID, и только те продукты, которые еще имеются на складе (см. критерий после ключевого слова WHERE), включаются в результирующий набор данных.

Листинг 4.1

```
CREATE VIEW "Products by Category" AS
  SELECT Categories.CategoryName, Products.ProductName,
         Products.QuantityPerUnit, Products.UnitsInStock,
         Products.Discontinued
         FROM Categories INNER JOIN Products ON Categories.CategoryID
 = Products.CategoryID
  WHERE Products.Discontinued <> 1
```

Теперь давайте создадим представление, показывающее все территории восточного региона. Это представление базируется на следующем запросе (листинг 4.2).

Листинг 4.2

```
SELECT Territories.TerritoryDescription,
Region.RegionDescription
  FROM Territories INNER JOIN
  Region ON Territories.RegionID = Region.RegionID
  WHERE Territories.RegionID = 1
```

Убедившись в том, что предложение SELECT возвращает результаты, которые нам нужны, мы добавляем оператор CREATE VIEW и присваиваем создаваемому представлению имя EASTTERR.

Вместо создания текста представления вручную можно использовать визуальные инструменты, обычно входящие в состав СУБД.

Для получения дополнительной информации о представлениях в Microsoft SQL Server мы можем использовать следующие системные хранимые процедуры:

- для получения сведений о представлении можно использовать системную хранимую процедуру sp_help. Например, sp_help EastTerr вернет сведения о только что созданном представлении;
- для получения исходного текста представления можно использовать хранимую процедуру sp_helptext;
- для того чтобы найти список таблиц, от которого зависит представление, можно использовать системную хранимую процедуру sp_depends;
- для переименования представления можно использовать системную хранимую процедуру sp_rename.

Мы рассмотрели, как использовать представления для получения данных, удовлетворяющих тем или иным критериям. Однако вернемся к последнему примеру. В базе данных NorthWind имеется четыре региона, и для получения списка территорий всех регионов нам нужно четыре разных представления. Эту задачу можно было бы упростить, если бы мы могли передать значение RegionID в качестве параметра. Это можно сделать с помощью хранимой процедуры.

4.3.2. Хранимые процедуры

Хранимая процедура - это скомпилированный набор SQL-предложений, сохраненный в базе данных как именованный объект и выполняющийся как единый фрагмент кода.

Хранимые процедуры могут принимать и возвращать параметры. Когда пользователь создает хранимую процедуру, сервер компилирует ее и помещает в разделяемый кэш, после чего скомпилированный код может быть применен несколькими пользователями. Когда приложение использует хранимую процедуру, оно передает ей параметры, если таковые требуются, и сервер выполняет процедуру без перекомпиляции.

Хранимые процедуры позволяют повысить производительность приложений. Во-первых, по сравнению с обычными SQL-запросами, посылаемыми из клиентского приложения, они требуют меньше времени для подготовки к выполнению, поскольку они уже скомпилированы и сохранены. Во-вторых, сетевой трафик в этом случае также меньше, чем в случае передачи SQL-запроса, так как по сети передается меньшее количество данных.

Хранимые процедуры автоматически перекомпилируются, если с объектами, на которые они влияют, произведены какие-либо изменения; иными словами, они всегда актуальны. Как уже было сказано выше, хранимые процедуры могут принимать параметры, что позволяет разным приложениям использовать одну и ту же процедуру, применяя различные наборы входных данных.

Хранимые процедуры обычно используются для поддержки ссылочной целостности данных и реализации бизнес-правил. В последнем случае достигается дополнительная гибкость, поскольку, если бизнес-правила изменяются, можно изменить только текст процедуры, не изменяя клиентских приложений.

Для создания, изменения и удаления процедур существуют специальные SQL-предложения - CREATE PROCEDURE, ALTER PROCEDURE и DROP PROCEDURE.

Предложение CREATE PROCEDURE.

Предложение CREATE PROCEDURE используется для создания хранимой процедуры. Оно имеет следующий упрощенный синтаксис:

```
CREATE PROC proc_name
[{@parameter data_type} [= default] [OUTPUT]]
[...]
AS
sql_statements
```

Аргумент proc_name устанавливает имя хранимой процедуры, которое должно быть уникально в рамках текущей базы данных. Аргумент @parameter определяет параметр процедуры. В предложении CREATE PROCEDURE можно определить один или более параметров. Если для параметра нет значения по умолчанию, он должен быть передан пользователем (или клиентским приложением) при вызове процедуры. В Microsoft SQL Server число параметров хранимой процедуры не должно превышать 1024; по умолчанию они могут иметь NULL-значения.

Некоторые универсальные механизмы доступа к данным могут накладывать дополнительные ограничения на число параметров хранимых процедур. Например, BDE-драйвер для Oracle 8 способен работать только с процедурами, число параметров которых не превышает 10.

Аргумент data_type указывает тип данных для параметра. Ключевое слово default может быть использовано для установки значений по умолчанию - это может быть константа или NULL. Если указано значение по умолчанию, процедура может быть вызвана без указания значения параметра. Если процедура использует параметр с ключевым словом LIKE, ее значение по умолчанию может содержать групповые символы (%, _, [] и [^]). Ключевое слово OUTPUT показывает, что это возвращаемый параметр.

Ключевое слово AS указывает действие, которое процедура должна выполнить, в виде любого количества SQL-предложений и предложений на процедурном расширении SQL, характерном для данного сервера.

Процедура, созданная с помощью предложения CREATE PROCEDURE, будет сохранена в текущей базе данных. В Microsoft SQL Server имена процедур содержатся в системной таблице sysobjects, а исходный текст - в таблице syscomments.

Предложение ALTER PROCEDURE.

Это SQL-предложение применяется для изменения уже существующей хранимой процедуры. Предложение ALTER PROCEDURE имеет тот же синтаксис, что и предложение CREATE PROCEDURE. Один из примеров, когда нужно изменить хранимую процедуру, - зашифровать ее исходный текст после того, как процедура отлажена, с целью скрыть ее исходный текст от других пользователей.

Предложение DROP PROCEDURE.

Это предложение используется для удаления хранимых процедур из базы данных. Предложение DROP PROCEDURE принимает один аргумент - имя удаляемой процедуры.

При удалении хранимой процедуры сведения о ней удаляются из системных таблиц sysobjects и syscomments.

Создание и использование хранимых процедур.

В разделе, посвященном представлениям, мы обращали внимание на то, что было бы удобно, если бы мы могли передать в представление параметр, содержащий значение RegionID для выбора одного из четырех регионов в базе данных NorthWind. Давайте еще раз рассмотрим запрос, возвращающий список территорий региона:

```
SELECT Territories.TerritoryDescription,
Region.RegionDescription
FROM Territories INNER JOIN
Region ON Territories.RegionID = Region.RegionID
WHERE Territories.RegionID = 1
```

Чтобы выбрать другой регион, нам нужно изменить условие в предложении WHERE в последней строке запроса. Следовательно, если мы используем переменную (назовем ее RegID), мы сможем выбрать один из четырех регионов без изменения других частей запроса.

В базе данных NorthWind четыре региона с номерами от 1 до 4. Это означает, что переменная RegID должна быть целого типа. Код хранимой процедуры приведен ниже:

```
CREATE PROCEDURE ShowRegion
@RegID int
AS
SELECT Territories.TerritoryDescription,
Region.RegionDescription
  FROM Territories INNER JOIN
  Region ON Territories.RegionID = Region.RegionID
  WHERE Territories.RegionID = @RegID
```

Обратите внимание на то, что мы оставили почти весь текст запроса SELECT нетронутым (он выделен курсивом) и только добавили предложение CREATE PROCEDURE с именем вновь созданной хранимой процедуры (в первой строке), объявление параметра (во второй строке) и ключевое слово AS, указывающее на начало предложений, реально выполняющих действия.

Очевидно, что мы можем применять хранимые процедуры не только для реализации расширенных версий представлений или «интеллектуальных» запросов SELECT. Хранимые процедуры предоставляют механизмы, позволяющие автоматизировать многие рутинные задачи.

В Microsoft SQL Server мы также можем использовать системные хранимые процедуры для работы с обычными хранимыми процедурами:

- sp_stored_procedures - показывает список хранимых процедур;
- sp_helptext - показывает исходный текст хранимой процедуры;
- sp_depends - показывает сведения о зависимостях хранимых процедур;
- sp_procoption - устанавливает опции хранимых процедур или устанавливает их;
- sp_recompile - перекомпилирует процедуру в момент ее следующего вызова;
- sp_rename - меняет имя процедуры.

Системные хранимые процедуры/

Поскольку мы говорим о Microsoft SQL Server, следует отметить огромное количество системных хранимых процедур, реализованных в нем. Имена системных хранимых процедур начинаются с SP_ или XP_ и хранятся в базе данных master. В табл. 4.9 приведен краткий список системных хранимых процедур.

Таблица 4.9

Системные хранимые процедуры

Системная хранимая процедура	Описание
sp_configure	Используется для вывода или изменения глобальных установок конфигурации сервера
sp_databases	Используется для получения списка баз данных, имеющихся на сервере или доступных с его помощью
sp_datatype_info	Используется для получения сведений о поддерживаемых типах данных
sp_help	Используется для получения сведений об объекте в базе данных
sp_helpdb	Используется для получения сведений об указанной базе данных или о всех базах данных, доступных с помощью данного сервера
sp_helpfile	Используется для получения сведений о физических именах и атрибутах файлов, в которых содержится текущая база данных
sp_monitor	Выводит статистику Microsoft SQL Server
xp_msver	Используется для получения сведений о версии сервера и связанных с этим данных

Системные хранимые процедуры можно использовать в клиентских приложениях. Для этого следует обратиться к базе данных master и выбрать имя соответствующей процедуры.

4.3.3. Триггеры

Триггер - это специальный тип хранимой процедуры, которая автоматически вызывается, когда данные в определенной таблице добавляются, удаляются или изменяются с помощью SQL-предложений INSERT, DELETE или UPDATE. В зависимости от того, какое из событий, связанных с изменением данных, инициирует запуск триггера, он называется insert trigger, delete trigger или update trigger.

Некоторые базы данных позволяют создавать разные триггеры для выполнения до и после вставки, удаления или изменения записей. Большинство СУБД позволяют создавать несколько триггеров для одного и того же события. В этом случае следует определить порядок, в котором они будут выполняться.

Триггеры часто используются для поддержки ссылочной целостности, каскадного изменения или удаления данных (то есть изменения или удаления всех дочерних записей вместе с родительской), архивирования удаленных или измененных данных, отслеживания изменений или вызова пользовательских или системных хранимых процедур.

Поскольку триггер вызывается автоматически самой СУБД (в нашем случае Microsoft SQL Server), его нельзя вызвать из клиентского приложения.

Триггеры могут опосредованно вызывать другие триггеры. Например, если триггер, выполняющийся в данный момент, содержит код, модифицирующий другую таблицу, имеющую собственный триггер, последний будет запущен. В свою очередь, он может вызвать следующий триггер и так далее. Такая последовательность триггеров называется термином nested triggers. Microsoft SQL Server поддерживает до 32 уровней вложения таких триггеров.

Как и многие другие объекты баз данных, триггеры создаются с помощью предложения CREATE. Чтобы изменить триггер, мы используем предложение ALTER, а для удаления триггера - предложение DROP.

Предложение CREATE TRIGGER.

Это предложение создает новый триггер для определенной таблицы для любого из предложений INSERT, DELETE или UPDATE. Ниже приведен упрощенный синтаксис такого предложения:

```
CREATE TRIGGER trigger_name
ON table_name
FOR {INSERT, UPDATE, DELETE}
[WITH ENCRYPTION]
AS
sql_statements
```

Аргумент trigger_name указывает имя триггера. Это имя должно быть уникально для базы данных. Аргумент table_name - имя таблицы, при модификации которой выполняется данный триггер. Таблица должна быть «настоящей» - нельзя указать имя представления вместо таблицы.

Ключевые слова {INSERT, UPDATE, DELETE} определяют, какие изменения данных активизируют триггер. Как минимум одно из ключевых слов этого списка обязательно должно присутствовать.

Необязательное ключевое слово [WITH ENCRYPTION] может быть использовано для сокрытия исходного кода триггера - в этом случае триггер не будет виден в таблице syscomments.

Ключевое слово AS указывает на действие, которое должен выполнить триггер, в виде любого количества SQL-предложений и предложений на процедурном расширении SQL, характерном для данного сервера. В них можно выполнять любые SQL-предложения, кроме перечисленных в приложении 4.

Триггеры не должны возвращать пользователю данные.

В предложении CREATE TRIGGER можно использовать две специальные таблицы. Например, таблицы deleted и inserted имеют ту же структуру, что и таблица, для которой определен триггер, и содержат старое и новое значения записей, измененных пользователем.

Например, мы можем использовать следующее SQL-предложение для поиска удаленных записей:

SELECT * FROM deleted

В табл. 4.10 показано содержимое таблиц deleted и inserted для всех возможных изменений данных.

Таблица 4.10

Возможные изменения данных

Изменение	Содержимое таблицы inserted	Содержимое таблицы deleted
INSERT	Добавленные записи	-
DELETE	-	Удаленные записи
UPDATE	Новые значения для модифицированных записей	Старые значения для модифицированных записей

Предложение ALTER TRIGGER.

Данное предложение имеет тот же синтаксис, что и предложение CREATE TRIGGER. Это предложение применяется для изменения текста имеющегося триггера. Например, если мы решили, что какой-либо триггер, определенный для всех трех типов модификаций данных, теперь не должен выполняться, когда записи удаляются, нам следует использовать тот же самый код, с помощью которого мы создавали триггер, но удалить ключевое слово UPDATE в предложении FOR.

Предложение DROP TRIGGER.

Это предложение удаляет триггер из базы данных. Используя это предложение, нужно указывать имя удаляемого триггера. Триггер автоматически удаляется при удалении таблицы, для которой он создан.

Создание и использование триггеров.

Существует много примеров различных триггеров, поддерживающих ссылочную целостность, выполняющих каскадное удаление и другие задачи. Мы создадим триггер, который будет использоваться для отслеживания изменений в таблице. Предположим, в компании имеется несколько менеджеров и каждый из них может добавлять, удалять и изменять данные о клиентах. Чтобы знать, кто и когда изменял эти данные, руководитель этих менеджеров должен иметь механизм для получения подобных сведений. Один их способов реализации такого механизма - создание триггера.

Для начала нам нужно добавить к таблице два новых поля, в которых будут содержаться эти сведения. Назовем их UpdatedBy (имя менеджера, обновившего запись последним) и UpdatedWhen (время, когда была изменена запись). Затем создадим триггер с именем KeepTrack. Вот его код:

```
CREATE TRIGGER KeepTrack ON Customers
FOR INSERT, UPDATE
AS
UPDATE Customers
SET Customers.UpdatedBy = USER_NAME(),
  Customers.UpdatedWhen = GETDATE()
```

FROM inserted, Customers

WHERE inserted.CustomerID = Customers.CustomerID

Как видно из исходного текста триггера, он выполняется после каждой операции INSERT и UPDATE в таблице Customers. Этот триггер будет сохранять имя менеджера (пользователя базы данных) в поле Customers.UpdatedBy, дату и время изменения - в поле Customers.UpdatedWhen. Эти данные извлекаются из временной таблицы inserted. Как видим, этот триггер позволяет следить за изменениями и вставкой новых записей в таблице.

Сведения об имеющихся триггерах можно найти в таблице sysobjects хранятся сведения о триггерах и их типах, а в таблице syscomments содержатся их исходные тексты.

ПРАКТИЧЕСКИЕ ЗАДАНИЯ

Практическое задание 4.1

Примеры запросов на выборку из следующих таблиц

РАСПИСАНИЕ

Номер группы	Номер подгруппы	Тип недели	День недели	Номер занятия	Корпус	Аудитория	Дисцилина	Вид занятий	Ф.И.О. преподавателя

СТУДЕНТ

Номер зачетной книжки	Группа	Ф.И.О.	Размер стипендии	Суммарный рейтинг

ДНИ НЕДЕЛИ

Номер дня недели	День недели

АУДИТОРИЯ

Корпус	Ауд.	Число мест	Сост. доски

НЕДЕЛИ

Дата пн.	Дата сб.	Номер недели	Тип недели

Задание 1. Выдать расписание по 2-м группам

Задание 2. Где сейчас преподаватель Иванов?

Задание 3. Где сейчас студент Иванов?

Задание 4. Найти аудиторию, достаточную для проведения собрания 100 человек 11.04.15.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

1. Дайте определение языка структурированных запросов.
2. Как происходит процесс обработки СУБД – запроса в SQL?
3. Дайте характеристику категорий операторов SQL.
4. Какой вид имеют операторы SQL?
5. Синтаксис оператора SELECT.
6. Когда используется предложение ORDER BY?
7. С помощью какой операции можно извлечь данные из нескольких таблиц?
8. Какие виды соединений вам известны?
9. Для чего можно использовать предложение GROUP BY?
10. Какие операторы SQL используются для обновления и удаления данных?
11. Какие операторы SQL используются для модификации метаданных?
12. Дайте определение объектов баз данных которые могут быть созданы и использованы с помощью средств SQL.
13. Что такое представление? Для чего используется?
14. Какие SQL-предложения используются для создания, модификации и удаления представлений?
15. Можно ли использовать запрос SELECT для создания представлений?
16. Что такое хранимая процедура? Для чего используется?
17. Что такое системная хранимая процедура? Когда используется?
18. Как называется специальный тип хранимой процедуры, которая автоматически вызывается, когда данные в определенной таблице добавляются, удаляются или изменяются с помощью SQL-предложений?
19. Должны ли триггеры возвращать пользователю данные?
20. Приведите примеры использования триггеров и хранимых процедур.

ГЛАВА 5. МЕТОДОЛОГИИ МОДЕЛИРОВАНИЯ ДАННЫХ

Моделирование данных представляет собой деятельность по обнаружению и документированию требований к информации. Требования к информации описывают данные и бизнес-правила, необходимые для поддержки предметной области. Модель данных может выражать как сложные информационные потребности целой корпорации, так и конкретные информационные потребности одной единственной программы. Моделирование данных обеспечивает наибольшую отдачу при применении на ранних стадиях жизненного цикла разработки баз данных. Модель предоставляет критически важную информацию для понимания функциональных границ проекта на итерационных фазах разработки. Начало фазы реализации без ясного понимания требований к данным может привести к тому, что ваш проект быстро выйдет за рамки отведенного бюджета или закончит свое существование на ряде недоделанных версий.

5.1. Структурное моделирование предметной области

Описанные модели предметной области нацелены на проектирование отдельных компонентов ИС: данных, функциональных программных модулей, управляющих программных модулей, программных модулей интерфейсов пользователей, структуры технического комплекса. Для более качественного проектирования указанных компонентов требуется построение моделей, увязывающих различные компоненты ИС между собой. В простейшем случае в качестве таких моделей взаимодействия могут использоваться матрицы перекрестных ссылок: «объекты-функции», «функции-события», «организационные единицы — функции», «организационные единицы — объекты», «организационные единицы — технические средства» и т д. Такие матрицы не наглядны и не отражают особенности реализации взаимодействий.

Для правильного отображения взаимодействий компонентов ИС важно осуществлять совместное моделирование таких компонентов, особенно с содержательной точки зрения объектов и функций. Методология структурного системного анализа существенно помогает в решении таких задач.

Структурным анализом принято называть метод исследования системы, которое начинается с ее общего обзора, а затем детализируется, приобретая иерархическую структуру с все большим числом уровней. Для таких методов характерно: разбиение на уровни абстракции с ограниченным числом элементов (от 3 до 6); ограниченный контекст, включающий только существенные детали каждого уровня; использование строгих формальных правил записи; последовательное приближение к результату. Структурный анализ основан на двух базовых принципах – «разделяй и властвуй» и принципе иерархической упорядоченности. Решение трудных проблем путем их разбиения на множество меньших независимых задач (так называемых «черных ящиков») и организация этих задач в древовидные иерархические структуры значительно повышают понимание сложных систем. Определим ключевые понятия структурного анализа.

Операция - элементарное (неделимое) действие, выполняемое на одном рабочем месте.

Функция - совокупность операций, сгруппированных по определенному признаку.

Бизнес-процесс - связанная совокупность функций, в ходе выполнения которой потребляются определенные ресурсы и создается продукт (предмет, услуга, научное открытие, идея), представляющая ценность для потребителя.

Подпроцесс - это бизнес-процесс, являющийся структурным элементом некоторого бизнес-процесса и представляющий ценность для потребителя.

Бизнес-модель - структурированное графическое описание сети процессов и операций, связанных с данными, документами, организационными единицами и прочими объектами, отражающими существующую или предполагаемую деятельность предприятия.

Существуют различные методологии структурного моделирования предметной области, среди которых следует выделить функционально-ориентированные и объектно-ориентированные методологии.

Целью методики является построение модели рассматриваемой системы в виде диаграммы потоков данных (Data Flow Diagram — DFD), обеспечивающей правильное описание выходов (отклика системы в виде данных) при заданном воздействии на вход системы (подаче сигналов через внешние интерфейсы). Диаграммы потоков данных являются основным средством моделирования функциональных требований к проектируемой системе.

При создании диаграммы потоков данных используются четыре основных понятия: потоки данных, процессы (работы) преобразования входных потоков данных в выходные, внешние сущности, накопители данных (хранилища).

Потоки данных являются абстракциями, использующимися для моделирования передачи информации (или физических компонент) из одной части системы в другую. Потоки на диаграммах изображаются именованными стрелками, ориентация которых указывает направление движения информации.

Назначение процесса (работы) состоит в продуцировании выходных потоков из входных в соответствии с действием, задаваемым именем процесса. Имя процесса должно содержать глагол в неопределенной форме с последующим дополнением (например, «получить документы по отгрузке продукции»). Каждый процесс имеет уникальный номер для ссылок на него внутри диаграммы, который может использоваться совместно с номером диаграммы для получения уникального индекса процесса во всей модели.

Хранилище (накопитель) данных позволяет на указанных участках определять данные, которые будут сохраняться в памяти между процессами. Фактически хранилище представляет «срезы» потоков данных во времени. Информация, которую оно содержит, может использоваться в любое время после ее получения, при этом данные могут выбираться в любом порядке. Имя хранилища должно определять его содержимое и быть существительным.

Внешняя сущность представляет собой материальный объект вне контекста системы, являющейся источником или приемником системных данных. Ее имя должно содержать существительное, например, «склад товаров». Предполагается, что объекты, представленные как внешние сущности, не должны участвовать ни в какой обработке.

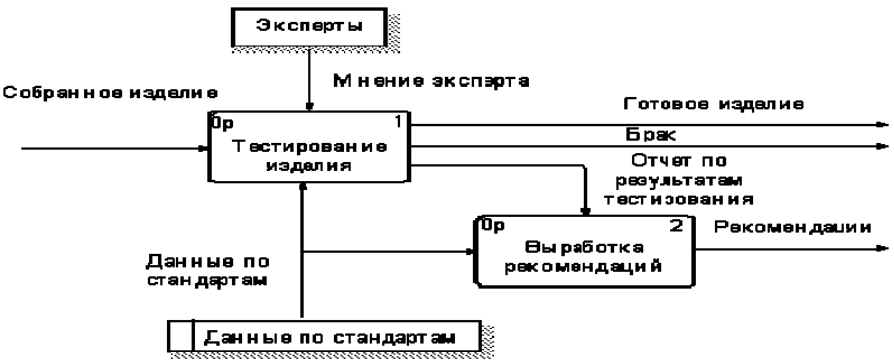

Рис.5.1. Пример DFD – диаграммы

Кроме основных элементов, в состав DFD входят словари данных и миниспецификации.

Словари данных являются каталогами всех элементов данных, присутствующих в DFD, включая групповые и индивидуальные потоки данных, хранилища и процессы, а также все их атрибуты.

Миниспецификации обработки — описывают DFD-процессы нижнего уровня. Фактически миниспецификации представляют собой алгоритмы описания задач, выполняемых процессами: множество всех миниспецификаций является полной спецификацией системы.

Процесс построения DFD начинается с создания так называемой основной контекстной диаграммы типа «звезда», на которой представлен моделируемый процесс и все внешние сущности, с которыми он взаимодействует. В случае сложного основного процесса он сразу представляется в виде декомпозиции на ряд взаимодействующих процессов. Критериями сложности в данном случае являются: наличие большого числа внешних сущностей, многофункциональность системы, ее распределенный характер. Внешние сущности выделяются по отношению к основному процессу. Для их определения необходимо выделить поставщиков и потребителей основного процесса, т.е. все объекты, которые взаимодействуют с основным процессом. На этом этапе описание взаимодействия заключается в выборе глагола, дающего представление о том, как внешняя сущность использует основной процесс или используется им. Например, основной процесс – «учет обращений граждан», внешняя сущность – «граждане», описание взаимодействия – «подает заявления и получает ответы». Этот этап является принципиально важным, поскольку именно он определяет границы моделируемой системы.

Для всех внешних сущностей строится таблица событий, описывающая их взаимодействие с основным потоком. Таблица событий включает в себя наименование внешней сущности, событие, его тип (типичный для системы или исключительный, реализующийся при определенных условиях) и реакцию системы.

На следующем шаге происходит декомпозиция основного процесса на набор взаимосвязанных процессов, обменивающихся потоками данных. Сами потоки не конкретизируются, определяется лишь характер взаимодействия. Декомпозиция завершается, когда процесс становится простым, т.е.:

1. процесс имеет два-три входных и выходных потока;
2. процесс может быть описан в виде преобразования входных данных в выходные;
3. процесс может быть описан в виде последовательного алгоритма.

Для простых процессов строится миниспецификация – формальное описание алгоритма преобразования входных данных в выходные.

Миниспецификация удовлетворяет следующим требованиям:

- для каждого процесса строится одна спецификация;
- спецификация однозначно определяет входные и выходные потоки для данного процесса;
- спецификация не определяет способ преобразования входных потоков в выходные;
- спецификация ссылается на имеющиеся элементы, не вводя новые;
- спецификация по возможности использует стандартные подходы и операции.

После декомпозиции основного процесса для каждого подпроцесса строится аналогичная таблица внутренних событий.

Следующим шагом после определения полной таблицы событий выделяются потоки данных, которыми обмениваются процессы и внешние сущности. Простейший способ их выделения заключается в анализе таблиц событий. События преобразуются в потоки данных от инициатора события к запрашиваемому процессу, а реакции – в обратный поток событий. После построения входных и выходных потоков аналогичным образом строятся внутренние потоки. Для их выделения для каждого из внутренних процессов выделяются поставщики и потребители информации. Если поставщик или потребитель информации представляет процесс сохранения или запроса информации, то вводится хранилище данных, для которого данный процесс является интерфейсом.

После построения потоков данных диаграмма должна быть проверена на полноту и непротиворечивость. Полнота диаграммы обеспечивается, если в системе нет «повисших» процессов, не используемых в процессе преобразования входных потоков в выходные. Непротиворечивость системы обеспечивается выполнением наборов формальных правил о возможных типах процессов:

- на диаграмме не может быть потока, связывающего две внешние сущности – это взаимодействие удаляется из рассмотрения;
- ни одна сущность не может непосредственно получать или отдавать информацию в хранилище данных – хранилище данных является пассивным элементом, управляемым с помощью интерфейсного процесса;
- два хранилища данных не могут непосредственно обмениваться информацией – эти хранилища должны быть объединены.

К преимуществам методики DFD относятся:

- возможность однозначно определить внешние сущности, анализируя потоки информации внутри и вне системы;
- возможность проектирования сверху вниз, что облегчает построение модели «как должно быть»;
- наличие спецификаций процессов нижнего уровня, что позволяет преодолеть логическую незавершенность функциональной модели и построить полную функциональную спецификацию разрабатываемой системы.

К недостаткам модели отнесем: необходимость искусственного ввода управляющих процессов, поскольку управляющие воздействия (потоки) и управляющие процессы с точки зрения DFD ничем не отличаются от обычных; отсутствие понятия времени, т.е. отсутствие анализа временных промежутков при преобразовании данных (все ограничения по времени должны быть введены в спецификациях процессов).

5.2. Методология визуального моделирования данных - IDEF1X

IDEF1X - высоко структурированная методология моделирования данных, расширяющая методологию IDEF1, принятую в качестве стандарта FIPS (Federal Information Processing Standards - федеральный орган стандартов обработки информации). IDEF1X использует строго структурированный набор типов конструкций моделирования и приводит к модели данных, которая требует понимания физической природы данных до того, как такая информация может стать доступной. ER-диаграммы были приняты в качестве основы для создания стандарта IDEF1X. Предварительный вариант этого стандарта был разработан в военно-воздушных силах США и предназначался для увеличения производительности при разработке компьютерных систем. В 1981 году этот стандарт был формализован и опубликован организацией ICAM (Integrated Computed Aided Manufacturing), и с тех пор является наиболее распространенным стандартом для создания моделей баз данных по всему миру.

IDEF1X является методом для разработки реляционных баз данных и использует условный синтаксис, специально разработанный для удобного построения концептуальной схемы.

Концептуальной схемой мы называем универсальное представление структуры данных в рамках коммерческого предприятия, независимое от конечной реализации базы данных и аппаратной платформы. Будучи статическим методом разработки, IDEF1X изначально не предназначен для динамического анализа по принципу "AS IS", тем не менее, он иногда применяется в этом качестве, как альтернатива методу IDEF1.

Использование метода IDEF1X наиболее целесообразно для построения логической структуры базы данных после того, как все информационные ресурсы исследованы и решение о внедрении реляционной базы данных, как части корпоративной информационной системы, было принято. Однако не стоит забывать, что средства моделирования IDEF1X специально разработаны для построения реляционных информационных систем, и если существует необходимость проектирования другой системы, скажем объектно-ориентированной, то лучше избрать другие методы моделирования.

Существует несколько очевидных причин, по которым IDEF1X не следует применять в случае построения нереляционных систем.

Во-первых, IDEF1X требует от проектировщика определить ключевые атрибуты, для того чтобы отличить одну сущность от другой, в то время как объектно-ориентированные системы не требуют задания ключевых ключей, в целях идентифицирования объектов.

Во-вторых, в тех случаях, когда более чем один атрибут является однозначно идентифицирующим сущность, проектировщик должен определить один из этих атрибутов первичным ключом, а все остальные вторичными. И, таким образом, построенная проектировщиком IDEF1X-модель, и переданная для окончательной реализации программисту является некорректной для применения методов объектно-ориентированной реализации, и предназначена для построения реляционной системы.

5.2.1. Концепция и семантика IDEF1X

Хотя терминология IDEF1X практически совпадает с терминологией IDEF1, существует ряд фундаментальных отличий в теоретических концепциях этих методологий.

Сущность в IDEF1X описывает собой совокупность или набор экземпляров похожих по свойствам, но однозначно отличаемых друг от друга по одному или нескольким признакам. Каждый экземпляр является реализацией сущности. Таким образом, сущность в IDEF1X описывает конкретный набор экземпляров реального мира, в отличие от сущности в IDEF1, которая представляет собой абстрактный набор информационных отображений реального мира. Примером сущности IDEF1X может быть сущность "СОТРУДНИК", которая представляет собой всех сотрудников предприятия, а один из них, скажем, Иванов Петр Сергеевич, является конкретной реализацией этой сущности. В примере, приведенном на рис. 5.2, каждый экземпляр сущности СОТРУДНИК содержит следующую информацию: ID сотрудника, имя сотрудника, адрес сотрудника и т.п. В IDEF1X модели эти свойства называются атрибутами сущности. Каждый атрибут содержит только часть информации о сущности.

Связи в IDEF1X представляют собой ссылки, соединения и ассоциации между сущностями. Связи это суть глаголы, которые показывают, как соотносятся сущности между собой. Ниже приведен ряд примеров связи между сущностями:

Отдел <состоит из> нескольких Сотрудников

Сотрудник <пишет> разные Отчеты.

Во всех перечисленных примерах взаимосвязи между сущностями соответствуют схеме один ко многим. Это означает, что один экземпляр первой сущности связан с несколькими экземплярами второй сущности. Причем первая сущность называется родительской, а вторая - дочерней. В приведенных примерах глаголы заключены в угловые скобки. Связи отображаются в виде линии между двумя сущностями с точкой на одном конце и глагольной фразой, отображаемой над линией. На рис.5.2 приводится диаграмма связи между Сотрудником и Отделом.

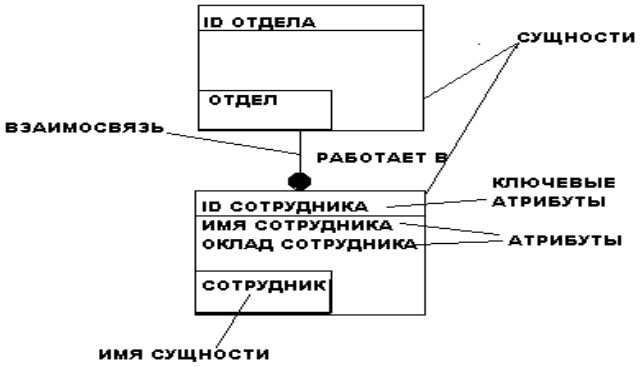

Рис.5.2. Диаграмма связи между сущностями Сотрудник и Отдел в IDEF1X

Отношения многие ко многим обычно используются на начальной стадии разработки диаграммы, например, в диаграмме зависимости сущностей и отображаются в IDEF1X в виде сплошной линии с точками на обоих концах. Так как отношения многие ко многим могут скрыть другие бизнес правила или ограничения, они должны быть полностью исследованы на одном из этапов моделирования. Например, иногда отношение многие ко многим на ранних стадиях моделирования идентифицируется неправильно, на самом деле представляя два или несколько случаев отношений один-ко-многим между связанными сущностями. Или, в случае необходимости хранения дополнительных сведений о связи многие-ко-многим, например, даты или комментария, такая связь должна быть заменена дополнительной сущностью, содержащей эти сведения. При моделировании необходимо быть уверенным в том, что все отношения многие ко многим будут подробно обсуждены на более поздних стадиях моделирования для обеспечения правильного моделирования отношений.

5.2.2 Идентификация сущностей. Представление о ключах

Сущность описывается в диаграмме IDEF1X графическим объектом в виде прямоугольника. На рис.5.3 приведен пример IDEF1X диаграммы. Каждый прямоугольник, отображающий собой сущность, разделяется горизонтальной линией на часть, в которой расположены ключевые поля и часть, где расположены неключевые поля. Верхняя часть называется ключевой областью, а нижняя часть областью данных. Ключевая область объекта СОТРУДНИК содержит поле "Уникальный идентификатор сотрудника", в области данных находятся поля "Имя сотрудника", "Адрес сотрудника", "Телефон сотрудника" и т.д.

Ключевая область содержит первичный ключ для сущности. Первичный ключ - это набор атрибутов, выбранных для идентификации уникальных экземпляров сущности. Атрибуты первичного ключа располагаются над линией в ключевой области. Как следует из названия, неключевой атрибут - это атрибут, который не был выбран ключевым. Неключевые атрибуты располагаются под чертой, в области данных. При создании сущности в IDEF1X модели, одним из главных вопросов, на который нужно ответить, является: "Как можно идентифицировать уникальную запись?". Для этого требуется уникальная идентификация каждой записи в сущности для того, чтобы правильно создать логическую

модель данных. Напомним, что сущности в IDEF1X всегда имеют ключевую область и, поэтому в каждой сущности должны быть определены ключевые атрибуты.

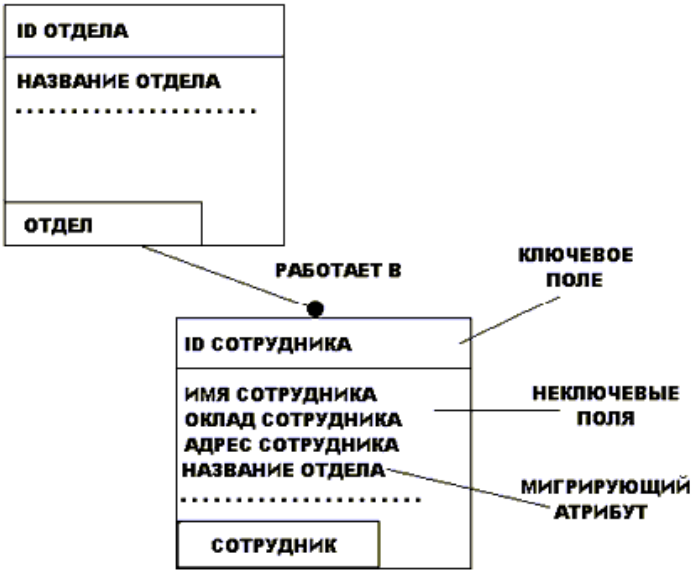

Рис.5.3. Пример IDEF1X - диаграммы

Выбор первичного ключа для сущности является очень важным шагом, и требует большого внимания. В качестве первичных ключей могут быть использованы несколько атрибутов или групп атрибутов. Атрибуты, которые могут быть выбраны первичными ключами, называются кандидатами в ключевые атрибуты (потенциальные атрибуты). Кандидаты в ключи должны уникально идентифицировать каждую запись сущности. В соответствии с этим, ни одна из частей ключа не может быть NULL, не заполненной или отсутствующей.

Например, для того, чтобы корректно использовать сущность СОТРУДНИК в IDEF1X модели данных (а позже в базе данных), необходимо иметь возможность уникально идентифицировать записи. Правила, по которым вы выбираете первичный ключ из списка предполагаемых ключей, очень строги, однако могут быть применены ко всем типам баз данных и информации. Правила устанавливают, что атрибуты и группы атрибутов должны:

- Уникальным образом идентифицировать экземпляр сущности.
- Не использовать NULL значений.
- Не изменяться со временем. Экземпляр идентифицируется при помощи ключа. При изменении ключа, соответственно меняется экземпляр.
- Быть как можно более короткими для использования индексирования и получения данных. Если вам нужно использовать ключ, являющийся комбинацией ключей из других сущностей, убедитесь в том, что каждая из частей ключа соответствует правилам.

Для наглядного представления о том, как целесообразно выбирать первичные ключи, приведем следующий пример - выберем первичный ключ для знакомой нам сущности "СОТРУДНИК":

- Атрибут "ID сотрудника" является потенциальным ключом, так как он уникален для всех экземпляров сущности СОТРУДНИК.
- Атрибут "Имя сотрудника" не очень хорош для потенциального ключа, так как среди служащих на предприятии может быть, к примеру, двое Иванов Петровых.
- Атрибут "Номер страхового полиса сотрудника" является уникальным, но проблема в том, что СОТРУДНИКА может не иметь такового.

- Комбинация атрибутов "имя сотрудника" и "дата рождения сотрудника" может оказаться удачной для наших целей и стать искомым потенциальным ключом.

После проведенного анализа можно назвать два потенциальных ключа - первый "Номер сотрудника" и комбинация, включающая поля "имя сотрудника" и "Дата рождения сотрудника". Так как атрибут "Номер сотрудника" имеет самые короткие и уникальные значения, то он лучше других подходит для первичного ключа.

При выборе первичного ключа для сущности, разработчики модели часто используют дополнительный (суррогатный) ключ, т.е. произвольный номер, который уникальным образом определяет запись в сущности. Атрибут "Номер сотрудника" является примером суррогатного ключа. Суррогатный ключ лучше всего подходит на роль первичного ключа потому, что является коротким и быстрее всего идентифицирует экземпляры в объекте. К тому же суррогатные ключи могут автоматически генерироваться системой так, чтобы нумерация была сплошной, т.е. без пропусков.

Потенциальные ключи, которые не выбраны первичными, могут быть использованы в качестве вторичных или альтернативных ключей. С помощью альтернативных ключей часто отображают различные индексы доступа к данным в конечной реализации реляционной базы.

Если сущности в IDEF1X диаграмме связаны, связь передает ключ (или набор ключевых атрибутов) дочерней сущности. Эти атрибуты называются внешними ключами. Внешние ключи определяются как атрибуты первичных ключей родительского объекта, переданные дочернему объекту через их связь. Передаваемые атрибуты называются мигрирующими.

5.2.3. Классификация сущностей в IDEF1X

При разработке модели, зачастую, приходится сталкиваться с сущностями, уникальность которых зависит от значений атрибута внешнего ключа. Для этих сущностей (для уникального определения каждой сущности) внешний ключ должен быть частью первичного ключа дочернего объекта.

Зависимой сущностью называется дочерняя сущность, уникальность которой зависит от атрибута внешнего ключа. В примере на рис.5.2 сущность СОТРУДНИК является зависимой сущностью потому, что его идентификация зависит от сущности ОТДЕЛ. В обозначениях IDEF1X зависимые сущности представлены в виде закругленных прямоугольников.

Зависимые сущности далее классифицируются на сущности, которые не могут существовать без родительской сущности и сущности, которые не могут быть идентифицированы без использования ключа родителя (сущности, зависящие от идентификации). Сущность СОТРУДНИК принадлежит ко второму типу зависимых сущностей, так как сотрудники могут существовать и без отдела.

Напротив, существуют ситуации, в которых сущность зависит от существования другой сущности. Рассмотрим две сущности: ЗАПРОС, используемый для отслеживания запросов покупателей, и ПОЗИЦИЯ ЗАПРОСА, который отслеживает отдельные элементы в ЗАПРОСе. Связь между этими двумя сущностями может быть выражена в виде ЗАПРОС <содержит> один или несколько ПОЗИЦИЙ ЗАПРОСА. В этом случае, ПОЗИЦИЯ ЗАПРОСА зависит от существования ЗАКАЗА.

Независимыми сущностями называются сущности, независящие при идентификации от других объектов в модели. В вышеописанном примере сущность ОТДЕЛ можно считать независимой. В IDEF1X независимые сущности представлены в виде прямоугольников.

В IDEF1X концепция зависимых и независимых сущностей усиливается типом взаимосвязей между двумя сущностями. Если вы хотите, чтобы внешний ключ передавался в дочернюю сущность (и, в результате, создавал зависимую сущность), то можете создать идентифицирующую связь между родительской и дочерней сущностями.

Идентифицирующие взаимосвязи обозначаются сплошной линией между сущностями.

Неидентифицирующие связи, являющиеся уникальными для IDEF1X, также связывают родительскую сущность с дочерней. Неидентифицирующие связи используются для отображения другого типа передачи атрибутов внешних ключей - передача в область данных дочерней сущности (под линией).

Неидентифицирующие связи отображаются пунктирной линией между объектами. Так как переданные ключи в неидентифицирующей связи не являются составной частью первичного ключа дочерней сущности, то этот вид связи не проявляется ни в одной идентифицирующей зависимости. В этом случае и ОТДЕЛ, и СОТРУДНИК рассматриваются как независимые сущности.

Тем не менее, взаимосвязь может отражать зависимость существования, если бизнес - правило для взаимосвязи определяет то, что внешний ключ не может принимать значение NULL. Если внешний ключ должен существовать, то это означает, что запись в дочерней сущности может существовать только при наличии ассоциированной с ним родительской записи.

Основным преимуществом IDEF1X, по сравнению с другими многочисленными методами разработки реляционных баз данных, является жесткая и строгая стандартизация моделирования. Установленные стандарты позволяют избежать различной трактовки построенной модели, которая, несомненно, является значительным недостатком ER.

5.3. Инструментарий визуального моделирования данных

С развитием компьютерных технологий и появлением CASE-моделирования (Computer Aided Software Engineering) возникла потребность в инструментах, которые бы поддерживали стандарты моделирования. Современный инструмент моделирования баз данных должен удовлетворять ряду требований.

- Позволять разработчику сконцентрироваться на самом моделировании, а не на проблемах с графическим отображением диаграммы. Инструмент должен автоматически размещать сущности на диаграмме, иметь развитые и простые в управлении средства визуализации и создания представлений модели.
- Инструмент должен проверять диаграмму на согласованность, автоматически определяя и разрешая несоответствия. Однако инструмент должен быть настраиваемым и при желании предоставлять разработчику некоторую свободу в действиях и право самому разрешать несоответствия или отступления от методологии.
- Инструмент моделирования должен поддерживать как логическое, так и физическое моделирование.
- Современный инструмент должен автоматически генерировать базу данных на СУБД назначения.

Все современные инструменты моделирования в той или иной степени удовлетворяют перечисленным выше общим требованиям.

5.3.1. Краткая характеристика AllFusion ERwin Data Modeler

AllFusion ERwin Data Modeler (ранее: ERwin) позволяет проектировать, документировать и сопровождать базы данных, хранилища данных и витрины данных (data marts). Создав наглядную модель базы данных, можно оптимизировать структуру БД и добиться её полного соответствия требованиям и задачам организации. Визуальное моделирование повышает качество создаваемой базы данных, продуктивность и скорость её разработки. Методологическую основу ERwin составляют технология IDEF1X и ER диаграммы, или диаграммы "сущность-связь". Пользователь описывает структуру данных визуально. Он задает служащие прообразами реляционных таблиц сущности с их атрибутами и при помощи мыши "натягивает" между ними связи, которые являются прототипами реляционных отношений.

ERwin не привязан к технологии какой-либо конкретной фирмы, поставляющей СУБД или средства разработки. Он поддерживает различные серверы баз данных и настольные СУБД, а также может обращаться к базе данных через ODBC.

AllFusion ERwin Data Modeler необходим всем компаниям, разрабатывающим и использующим базы данных, администраторам баз данных, системным аналитикам, проектировщикам БД, разработчикам, руководителям проектов. Для доказательства приведем некоторые аргументы и факты:

- поддерживается прямое (создание БД на основе модели) и обратное (генерация модели по имеющейся базе данных) проектирование для более двух десятков типов СУБД;
- увеличивает производительность труда благодаря удобному интерфейсу и автоматизации рутинных процедур;
- поддерживает методологию структурного моделирования SADT и следующие нотации: IDEF1x, IE, Dimensional (последняя - для проектирования хранилищ данных);
- ERwin является стандартом де-факто;
- поддерживает два десятка различных СУБД: настольные, реляционные и специализированные СУБД, предназначенные для создания хранилищ данных;
- интегрирован с линейкой продуктов Computer Associates для поддержки всех стадий разработки ИС, CASE-средствами Oracle Designer, Rational Rose, средствами разработки и др.;
- позволяет повторно использовать компоненты созданных ранее моделей, а также использовать наработки других разработчиков. Повышается эффективность!;
- возможна совместная работа группы проектировщиков с одними и теми же моделями (с помощью AllFusion Model Manager);
- позволяет переносить структуру БД (не сами данные!) из СУБД одного типа в другой;
- позволяет документировать структуру БД;
- продукт легко освоить;
- в России тысячи пользователей ERwin. Так, книга по ERwin и BPwin Сергея Маклакова, руководителя УЦ Interface Ltd., разошлась тиражом 15 тыс.;
- продукт можно использовать на всех стадиях жизненного цикла баз данных: при проектировании, разработке, тестировании и поддержке.

ERwin предназначен для разработчиков, проектировщиков БД, системных аналитиков. С помощью ERwin разработчик может сначала, используя визуальные средства, описать схему БД, а затем автоматически сгенерировать файлы данных для выбранной реляционной СУБД. Автоматически генерируются также триггеры, обеспечивающие

ссылочную целостность БД. Поддерживаются хранимые процедуры. Возможна также обратная разработка - восстановление модели данных по имеющимся файлам БД.

Существует несколько методологий визуального моделирования данных. ERwin поддерживает две:

- IDEF1X (Integration Definition for Information Modeling - интегрированное описание информационных моделей).
- IE (Information Engineering - информационная инженерия).

ERwin имеет два уровня представления модели - логический и физический. Логический уровень - это абстрактный взгляд на данные, когда данные представляются так, как выглядят в реальном мире, и могут называться так, как они называются в реальном мире, например "Постоянный клиент", "Отдел" или "Фамилия сотрудника". Объекты модели, представляемые на логическом уровне, называются сущностями и атрибутами. Логическая модель данных может быть построена на основе другой логической модели, например на основе модели процессов. Логическая модель данных является универсальной и никак не связана с конкретной реализацией СУБД.

Физическая модель данных, напротив, зависит от конкретной СУБД, фактически являясь отображением системного каталога. В физической модели содержится информация обо всех объектах БД. Поскольку стандартов на объекты БД не существует (например, нет стандарта на типы данных), физическая модель зависит от конкретной реализации СУБД. Следовательно, одной и той же логической модели могут соответствовать несколько разных физических моделей. Если в логической модели не имеет значения, какой конкретно тип данных имеет атрибут, то в физической модели важно описать всю информацию о конкретных физических объектах - таблицах, колонках, индексах, процедурах и т.д.

Документирование модели

Многие СУБД имеют ограничение на именование объектов (например, ограничение на длину имени таблицы или запрет использования специальных символов — пробела и т. п.). Зачастую разработчики ИС имеют дело с нелокализованными версиями СУБД. Это означает, что объекты БД могут называться короткими словами, только латинскими символами и без использования специальных символов (т. е. нельзя назвать таблицу, используя предложение — ее можно назвать только одним словом). Кроме того, проектировщики БД нередко злоупотребляют "техническими" наименованиями, в результате таблица и колонки получают наименования типа RTD_324 или CUST_A12 и т.д. Полученную в результате структуру могут понять только специалисты (а чаще всего — только авторы модели), ее невозможно обсуждать с экспертами предметной области. Разделение модели на логическую и физическую позволяет решить эту проблему. На физическом уровне объекты БД могут называться так, как того требуют ограничения СУБД. На логическом уровне можно этим объектам дать синонимы — имена более понятные неспециалистам, в том числе на кириллице и с использованием специальных символов. Например, таблице CUST_A12 может соответствовать сущность Постоянный клиент. Такое соответствие позволяет лучше документировать модель и дает возможность обсуждать структуру данных с экспертами предметной области.

Масштабирование

Создание модели данных, как правило, начинается с разработки логической модели. После описания логической модели проектировщик может выбрать необходимую СУБД, и ERwin автоматически создаст соответствующую физическую модель. На основе физической модели ERwin может сгенерировать системный каталог СУБД или соответствующий SQL-скрипт. Этот процесс называется прямым проектированием (Forward Engineering). Тем самым достигается масштабируемость — создав одну логическую модель данных, можно сгенерировать физические модели под любую поддерживаемую ERwin СУБД. С другой стороны, ERwin способен по содержимому системного каталога или SQL-скрипту воссоздать

физическую и логическую модель данных (Reverse Engineering). На основе полученной логической модели данных можно сгенерировать физическую модель для другой СУБД и затем создать ее системный каталог. Следовательно, ERwin позволяет решить задачу по переносу структуры данных с одного сервера на другой. Например, можно перенести структуру данных с Oracle на Informix (или наоборот) или перенести структуру dbf-файлов в реляционную СУБД, тем самым облегчив переход от файл-серверной к клиент-серверной ИС. Однако, формальный перенос структуры "плоских" таблиц на реляционную СУБД обычно неэффективен. Для того чтобы извлечь выгоды от перехода на клиент-серверную технологию, структуру данных следует модифицировать.

Для переключения между логической и физической моделью данных служит список выбора в центральной части панели инструментов ERwin (рис.5.4).

Если при переключении физической модели еще не существует, она будет создана автоматически.

Рис. 5.4. Переключение между логической и физической моделью

Интерфейс ERwin. Уровни отображения модели.

Интерфейс выполнен в стиле Windows-приложений, достаточно прост и интуитивно понятен. Рассмотрим кратко основные функции ERwin по отображению модели.

Каждому уровню отображения модели соответствует своя палитра инструментов. На логическом уровне палитра инструментов имеет следующие кнопки:

- кнопку указателя (режим мыши) — в этом режиме можно установить фокус на каком-либо объекте модели;
- кнопку внесения сущности;
- кнопку категории (категория, или категориальная связь, — специальный тип связи между сущностями, которая будет рассмотрена ниже);
- кнопку внесения текстового блока;
- кнопку перенесения атрибутов внутри сущностей и между ними;
- кнопки создания связей: идентифицирующую, "многие-ко-многим" и неидентифицирующую.

На физическом уровне палитра инструментов имеет:

- вместо кнопки категорий — кнопку внесения представлений (view);
- вместо кнопки связи "многие-ко-многим" — кнопку связей представлений.

Для создания моделей данных в ERwin можно использовать две нотации: IDEFIX и IE (Information Engineering). В дальнейшем будет рассматриваться нотация IDEFIX.

ERwin имеет несколько уровней отображения диаграммы: уровень сущностей, уровень атрибутов, уровень определений, уровень первичных ключей и уровень иконок. Переключиться между первыми тремя уровнями можно с использованием кнопок панели инструментов. Переключиться на другие уровни отображения можно при помощи контекстного меню, которое появляется, если "кликнуть" по любому месту диаграммы, не занятому объектами модели. В контекстном меню следует выбрать пункт Display Level и затем — необходимый уровень отображения.

5.3.2. Создание логической модели данных

Уровни логической модели. Различают три уровня логической модели, отличающихся по глубине представления информации о данных:

- диаграмма сущность-связь (Entity Relationship Diagram, ERD);
- модель данных, основанная на ключах (Key Based model, KB);
- полная атрибутивная модель (Fully Attributed model, FA).

Диаграмма сущность-связь представляет собой модель данных верхнего уровня. Она включает сущности и взаимосвязи, отражающие основные бизнес-правила предметной области. Такая диаграмма не слишком детализирована, в нее включаются основные сущности и связи между ними, которые удовлетворяют основным требованиям, предъявляемым к ИС. Диаграмма сущность-связь может включать связи "многие-ко-многим" и не включать описание ключей. Как правило, ERD используется для презентаций и обсуждения структуры данных с экспертами предметной области.

Модель данных, основанная на ключах, — более подробное представление данных. Она включает описание всех сущностей и первичных ключей и предназначена для представления структуры данных и ключей, которые соответствуют предметной области.

Полная атрибутивная модель — наиболее детальное представление структуры данных: представляет данные в третьей нормальной форме и включает все сущности, атрибуты и связи.

Основные компоненты диаграммы ERwin — это сущности, атрибуты и связи. Каждая сущность является множеством подобных индивидуальных объектов, называемых экземплярами. Каждый экземпляр индивидуален и должен отличаться от всех остальных экземпляров. Атрибут выражает определенное свойство объекта. С точки зрения БД (физическая модель) сущности соответствует таблица, экземпляру сущности — строка в таблице, а атрибуту — колонка таблицы.

Построение модели данных предполагает определение сущностей и атрибутов, т. е. необходимо определить, какая информация будет храниться в конкретной сущности или атрибуте.

Сущность можно определить как объект, событие или концепцию, информация о которых должна сохраняться. сущности должны иметь наименование с четким смысловым значением, именоваться существительным в единственном числе, не носить "технических" наименований и быть достаточно важными для того, чтобы их моделировать. Именование сущности в единственном числе облегчает в дальнейшем чтение модели. Фактически имя сущности дается по имени ее экземпляра. Примером может быть сущности Заказчик (но не Заказчики!) с атрибутами Номер заказчика, Фамилия заказчика и Адрес заказчика. На уровне физической модели ей может соответствовать таблица Customer с колонками Customer_number, Customer_name и Customer_address. Каждая сущность должна быть полностью определена с помощью текстового описания. Для внесения дополнительных комментариев и определений к сущности служат свойства, определенные пользователем (UDP). Использование (UDP) аналогично их использованию в BPwin.

Как было указано выше, каждый атрибут хранит информацию об определенном свойстве сущности, а каждый экземпляр сущности должен быть уникальным. Атрибут или группа атрибутов, которые идентифицируют сущность, называется первичным ключем.

Атрибуты должны именоваться в единственном числе и иметь четкое смысловое значение. Соблюдение этого правила позволяет частично решить проблему нормализации данных уже на этапе определения атрибутов. Например, создание в сущности Сотрудник атрибута Телефоны сотрудника противоречит требованиям нормализации, поскольку атрибут должен быть атомарным, т.е. не содержать множественных значений. Согласно

синтаксису IDEFIX имя атрибута должно быть уникально в рамках модели (а не только в рамках сущности!). По умолчанию при попытке внесения уже существующего имени атрибута ERwin переименовывает его.

Каждый атрибут должен быть определен, при этом следует избегать циклических определений, например, когда термин 1 определяется через термин 2, термин 2 — через термин 3, а термин 3 в свою очередь — через термин 1. Часто приходится создавать производные атрибуты, т. е. атрибуты, значение которых можно вычислить из других атрибутов. Примером производного атрибута может служить Возраст сотрудника, который может быть вычислен из атрибута Дата рождения сотрудника. Такой атрибут может привести к конфликтам; действительно, если вовремя не обновить значение атрибута Возраст сотрудника, он может противоречить значению атрибута Дата рождения сотрудника. Производные атрибуты — ошибка нормализации, однако их вводят для повышения производительности системы, чтобы не проводить вычисления, которые на практике могут быть сложными.

Связь является логическим соотношением между сущностями. Каждая связь должна именоваться глаголом или глагольной фразой. Имя связи выражает некоторое ограничение или бизнес-правило и облегчает чтение диаграммы. По умолчанию имя связи на диаграмме не показывается. На логическом уровне можно установить идентифицирующую связь "один-ко-многим", связь "многие-ко-многим" и неидентифицирующую связь "один-ко-многим".

В IDEFIX различают зависимые и независимые сущности. Тип сущности определяется ее связью с другими сущностями. Идентифицирующая связь устанавливается между независимой (родительский конец связи) и зависимой (дочерний конец связи) сущностями. Когда рисуется идентифицирующая связь, ERwin автоматически преобразует дочернюю сущность в зависимую. Зависимая сущность изображается прямоугольником со скругленными углами. Экземпляр зависимой сущности определяется только через отношение к родительской сущности. При установлении идентифицирующей связи атрибуты первичного ключа родительской сущности автоматически переносятся в состав первичного ключа дочерней сущности. Эта операция дополнения атрибутов дочерней сущности при создании связи называется миграцией атрибутов. В дочерней сущности новые атрибуты помечаются как внешний ключ — FK.

При установлении неидентифицирующей связи дочерняя сущность остается независимой, а атрибуты первичного ключа родительской сущности мигрируют в состав неключевых компонентов родительской сущности. Неидентифицирующая связь служит для связывания независимых сущностей.

Идентифицирующая связь показывается на диаграмме сплошной линией с жирной точкой на дочернем конце связи, неидентифицирующая – пунктирной (рис.5.5).

Мощность связей (Cardinality) — служит для обозначения отношения числа экземпляров родительской сущности к числу экземпляров дочерней.

Различают четыре типа мощности:

- общий случай, когда одному экземпляру родительской сущности соответствуют 0, 1 или много экземпляров дочерней сущности; не помечается каким-либо символом;
- символом P помечается случай, когда одному экземпляру родительской сущности соответствуют 1 или много экземпляров дочерней сущности (исключено нулевое значение);
- символом Z помечается случай, когда одному экземпляру родительской сущности соответствуют 0 или 1 экземпляр дочерней сущности (исключены множественные значения);

- цифрой помечается случай точного соответствия, когда одному экземпляру родительской сущности соответствует заранее заданное число экземпляров дочерней сущности.

Имя связи (Verb Phrase) — фраза, характеризующая отношение между родительской и дочерней сущностями. Для связи "один-ко-многим", идентифицирующей или неидентифицирующей, достаточно указать имя, характеризующее отношение от родительской к дочерней сущности (Parent-to-Child). Для связи многие-ко-многим следует указывать имена как Parent-to-Child, так и Child-to-Parent.

Типы сущностей и иерархия наследования. Как было указано выше, связи определяют, является ли сущность независимой или зависимой. Различают несколько типов зависимых сущностей.

Характеристическая — зависимая дочерняя сущность, которая связана только с одной родительской и по смыслу хранит информацию о характеристиках родительской сущности (рис. 5.5).

Рис. 5.5. Пример характеристической сущности "Хобби"

Ассоциативная — сущность, связанная с несколькими родительскими сущностями. Такая сущность содержит информацию о связях сущностей.

Именующая — частный случай ассоциативной сущности, не имеющей собственных атрибутов (только атрибуты родительских сущностей, мигрировавших в качестве внешнего ключа).

Категориальная — дочерняя сущность в иерархии наследования.

Иерархия наследования (или иерархия категорий) представляет собой особый тип объединения сущностей, которые разделяют общие характеристики. Например, в организации работают служащие, занятые полный рабочий день (постоянные служащие), и совместители. Из их общих свойств можно сформировать обобщенную сущность (родовой предок) Сотрудник (рис. 5.6), чтобы представить информацию, общую для всех типов служащих. Специфическая для каждого типа информация может быть расположена в категориальных сущностях (потомках) Постоянный сотрудник и Совместитель.

Рис. 5.6. Иерархия наследования. Неполная категория

Обычно иерархию наследования создают, когда несколько сущностей имеют общие по смыслу атрибуты, либо когда сущности имеют общие по смыслу связи (например, если бы Постоянный сотрудник и Совместитель имели сходную по смыслу связь "работает в" с сущностью Организация), либо когда это диктуется бизнес-правилами. Для каждой

категории можно указать дискриминатор — атрибут родового предка, который показывает, как отличить одну категориальную сущность от другой (атрибут Тип на рис.5.6).

Иерархии категорий делятся на два типа — полные и неполные. В полной категории одному экземпляру родового предка (сущность Служащий, рис. 5.7) обязательно соответствует экземпляр в каком-либо потомке, т. е. в примере служащий обязательно является либо совместителем, либо консультантом, либо постоянным сотрудником.

Рис. 5.7. Иерархия наследования. Полная категория

Если категория еще не выстроена полностью и в родовом предке могут существовать экземпляры, которые не имеют соответствующих экземпляров в потомках, то такая категория будет неполной. На рис. 5.6 показана неполная категория — сотрудник может быть не только постоянным или совместителем, но и консультантом, однако сущность Консультант еще не внесена в иерархию наследования.

Ключи. Как было сказано выше, каждый экземпляр сущности должен быть уникален и должен отличаться от других атрибутов.

Первичный ключ (primary key) — это атрибут или группа атрибутов, однозначно идентифицирующая экземпляр сущности, атрибуты первичного ключа на диаграмме не требуют специального обозначения — это те атрибуты, которые находятся в списке атрибутов выше горизонтальной линии (рис. 5.7).

В одной сущности могут оказаться несколько атрибутов или наборов атрибутов, претендующих на роль первичного ключа. Такие претенденты называются потенциальными ключами (candidate key).

Ключи могут быть сложными, т. е. содержащими несколько атрибутов. Сложные первичные ключи не требуют специального обозначения — это список атрибутов, расположенных выше горизонтальной линии.

Рассмотрим кандидатов на роль первичного ключа сущности Сотрудник (рис. 5.8).

Здесь можно выделить следующие потенциальные ключи:

1. Табельный номер;
2. Номер паспорта;
3. Фамилия + Имя + Отчество.

Для того чтобы стать первичным, потенциальный ключ должен удовлетворять ряду требований:

Уникальность. Два экземпляра не должны иметь одинаковых значений возможного ключа, потенциальный ключ № 3 (Фамилия + Имя + Отчество) является плохим кандидатом, поскольку в организации могут работать полные тезки.

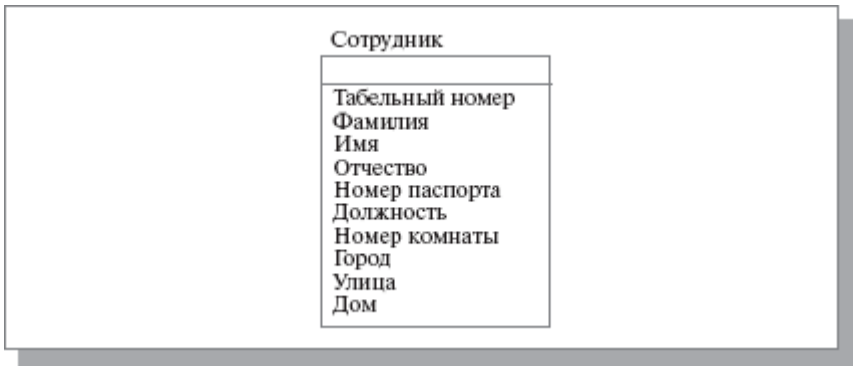

Рис. 5.8. Определение первичного ключа для сущности "Сотрудник"

Компактность. Сложный возможный ключ не должен содержать ни одного атрибута, удаление которого не приводило бы к утрате уникальности. Для обеспечения уникальности ключа № 3 дополним его атрибутами Дата рождения и Цвет волос. Если бизнес-правила говорят, что сочетания атрибутов Фамилия + Имя + Отчество + Дата рождения достаточно для однозначной идентификации сотрудника, то Цвет волос оказывается лишним, т. е. ключ Фамилия + Имя + Отчество + Дата рождения + Цвет волос не является компактным.

При выборе первичного ключа предпочтение должно отдаваться более простым ключам, т. е. ключам, содержащим меньшее количество атрибутов. В приведенном примере ключи № 1 и 2 предпочтительней ключа № 3.

Атрибуты ключа не должны содержать нулевых значений. Значение атрибутов ключа не должно меняться в течение всего времени существования экземпляра сущности. Сотрудница организации может выйти замуж и сменить как фамилию, так и паспорт. Поэтому ключи № 2 и 3 не подходят на роль первичного ключа.

Каждая сущность должна иметь по крайней мере один потенциальный ключ. Многие сущности имеют только один потенциальный ключ. Такой ключ становится первичным. Некоторые сущности могут иметь более одного возможного ключа. Тогда один из них становится первичным, а остальные — альтернативными ключами.

Альтернативный ключ (Alternate Key) — это потенциальный ключ, не ставший первичным.

Нормализация данных.

Нормализация данных — процесс проверки и реорганизации сущностей и атрибутов с целью удовлетворения требований к реляционной модели данных. Нормализация позволяет быть уверенным, что каждый атрибут определен для своей сущности, а также значительно сократить объем памяти для хранения информации и устранить аномалии в организации хранения данных. В результате проведения нормализации должна быть создана структура данных, при которой информация о каждом факте хранится только в одном месте. Процесс нормализации сводится к последовательному приведению структуры данных к нормальным формам — формализованным требованиям к организации данных. Известны шесть нормальных форм.

На практике обычно ограничиваются приведением данных к третьей нормальной форме. Для углубленного изучения нормализации рекомендуется книга К. Дж. Дейта "Введение в системы баз данных" (Киев; М.: Диалектика, 1998).

ERwin не содержит полного алгоритма нормализации и не может проводить нормализацию автоматически, однако его возможности облегчают создание

нормализованной модели данных. Запрет на присвоение неуникальных имен атрибутов в рамках модели (при соответствующей установке опции Unique Name) облегчает соблюдение правила "один факт — в одном месте". Имена ролей атрибутов внешних ключей и унификация атрибутов также облегчают построение нормализованной модели.

Домены. Домен можно определить как совокупность значений, из которых берутся значения атрибутов. Каждый атрибут может быть определен только на одном домене, но на каждом домене может быть определено множество атрибутов. В понятие домена входит не только тип данных, но и область значений данных. Например, можно определить домен "Возраст" как положительное целое число и определить атрибут Возраст сотрудника как принадлежащий этому домену.

В ERwin домен может быть определен только один раз и использоваться как в логической, так и в физической модели. Домены позволяют облегчить работу с данными как разработчикам на этапе проектирования, так и администраторам БД на этапе эксплуатации системы. На логическом уровне домены можно описать без конкретных физических свойств. На физическом уровне они автоматически получают специфические свойства, которые можно изменить вручную. Так, домен "Возраст" может иметь на логическом уровне тип Number, на физическом уровне колонкам домена будет присвоен тип INTEGER.

Каждый домен может быть описан, снабжен комментарием или свойством, определенным пользователем (UDP).

5.3.3. Создание физической модели данных

Физическая модель содержит всю информацию, необходимую для реализации конкретной БД. Различают два уровня физической модели:

- трансформационную модель;
- модель СУБД.

Трансформационная модель содержит информацию для реализации отдельного проекта, который может быть частью общей ИС и описывать подмножество предметной области. Данная модель позволяет проектировщикам и администраторам БД лучше представить, какие объекты БД хранятся в словаре данных, и проверить, насколько физическая модель удовлетворяет требованиям к ИС.

Модель СУБД автоматически генерируется из трансформационной модели и является точным отображением системного каталога СУБД.

Физический уровень представления модели зависит от выбранного сервера баз данных. ERwin поддерживает более 20 реляционных и нереляционных БД.

По умолчанию ERwin генерирует имена таблиц и индексов по шаблону на основе имен соответствующих сущностей и ключей логической модели, которые в дальнейшем могут быть откорректированы вручную. Имена таблиц и колонок будут сгенерированы по умолчанию на основе имен сущностей и атрибутов логической модели.

Правила валидации и значения по умолчанию. ERwin поддерживает правила валидации для колонок, а также значение, присваиваемое колонкам по умолчанию. Правило валидации задает список допустимых значений для конкретной колонки и/или правила проверки допустимых значений. В список допустимых значений можно вносить новые значения. ERwin позволяет сгенерировать правила валидации соответственно синтаксису выбранной СУБД с учетом границ диапазона или списка значений.

Значение по умолчанию – значение, которое нужно ввести в колонку, если никакое другое значение не задано явным образом во время ввода данных. С каждой колонкой или доменом можно связать значение по умолчанию. Список значений можно редактировать.

После создания правила валидации и значения по умолчанию их можно присвоить одной или нескольким колонкам или доменами.

Индексы. В БД данные обычно хранятся в том порядке, в котором их ввели в таблицу. Многие реляционные СУБД имеют страничную организацию, при которой таблица может храниться фрагментарно в разных областях диска, причем строки таблицы располагаются на страницах неупорядоченно. Такой способ позволяет быстро вводить новые данные, но затрудняет поиск данных. Чтобы решить проблему поиска, СУБД используют объекты, называемые индексами.

Индекс содержит отсортированную по колонке или нескольким колонкам информацию и указывает на строки, в которых хранится конкретное значение колонки. Поскольку значения в индексе хранятся в определенном порядке, при поиске просматривать нужно значительно меньший объем данных, что существенно уменьшает время выполнения запроса. Индекс рекомендуется создавать для тех колонок, по которым часто производится поиск.

При генерации схемы физической БД ERwin автоматически создает индекс на основе первичного ключа каждой таблицы, а также на основе всех альтернативных ключей и внешних ключей, поскольку эти колонки наиболее часто используются для поиска данных. Можно отказаться от генерации индексов по умолчанию и создать собственные индексы. Для увеличения эффективности поиска администратор БД должен анализировать часто выполняемые запросы и на основе анализа создавать собственные индексы.

5.3.4. Триггеры и хранимые процедуры

Триггеры и хранимые процедуры – это именованные блоки кода SQL, которые заранее откомпилированы и хранятся на сервере для того, чтобы быстро производить обработку запросов, валидацию данных и другие часто выполняемые функции. Хранение и выполнение кода на сервере позволяет создавать код только один раз, а не в каждом приложении, работающем с БД. Это экономит время при написании и сопровождении программ. При этом гарантируется, что целостность данных и бизнес-правила поддерживаются независимо от того, какое именно клиентское приложение обращается к данным. Триггеры и хранимые процедуры не требуется пересылать по сети из клиентского приложения, что значительно снижает сетевой трафик. Хранимой процедурой называется именованный набор предварительно откомпилированных команд SQL, который может вызываться из клиентского приложения или из другой хранимой процедуры.

Триггером называется процедура, которая выполняется автоматически как реакция на событие. Таким событием может быть вставка, изменение или удаление строки в существующей таблице. Триггер сообщает СУБД, какие действия нужно выполнить при выполнении команд SQL INSERT, UPDATE или DELETE для обеспечения дополнительной функциональности, выполняемой на сервере.

Триггер ссылочной целостности – это особый вид триггера, используемый для поддержания целостности между двумя таблицами, которые связаны между собой. Если строка в одной таблице вставляется, изменяется или удаляется, то триггер ссылочной целостности сообщает СУБД, что нужно делать с теми строками в других таблицах, у которых значение внешнего ключа совпадает со значением первичного ключа вставленной строки (измененной или удаленной строки).

Для генерации триггеров ERwin использует механизм шаблонов – специальных скриптов, использующих макрокоманды. При генерации кода триггера вместо макрокоманд подставляются имена таблиц, колонок, переменные и другие фрагменты кода,

соответствующие синтаксису выбранной СУБД. Шаблоны триггеров ссылочной целостности, генерируемые ERwin по умолчанию, можно изменять.

Для создания и редактирования хранимых процедур ERwin располагает специальными редакторами, аналогичными редакторам, используемым для создания триггеров. В отличие от триггера хранимая процедура не выполняется в ответ на какое-то событие, а вызывается из другой программы, которая передает на сервер имя процедуры. Хранимая процедура более гибкая, чем триггер, поскольку может вызывать другие хранимые процедуры. Ей можно передавать параметры, и она может возвращать параметры, значения и сообщения.

Создание отчетов

Для генерации отчетов в ERwin имеется простой и эффективный инструмент – Report Browser. По умолчанию Report Browser содержит предварительно определенные отчеты, позволяющие наглядно представить информацию об основных объектах модели данных – как логической, так и физической. С помощью специального редактора существующие отчеты можно изменить или создать собственный отчет. Каждый отчет может быть настроен индивидуально, данные в нем могут быть отсортированы и отфильтрованы. Browser Report позволяет сохранять результаты выполнения отчетов, печатать и экспортировать их в распространенные форматы.

Генерация словарей

Для управления большими проектами ERwin имеет специальный инструмент – ERwin Dictionary, который обеспечивает коллективную работу над диаграммами и позволяет сохранять и документировать различные версии моделей данных. ERwin Dictionary представляет собой специальную БД, которая позволяет решить проблемы документирования и хранения моделей, однако не полностью отвечает требованиям многопользовательской работы.

5.4. Объектный подход к моделированию данных с использованием языка UML

Унифицированный язык моделирования (Unified Modeling Language - UML) - это язык для специфицирования, визуализации, конструирования и документирования на основе объектно-ориентированного подхода разных видов систем: программных, аппаратных, программно-аппаратных, смешанных, явно включающие деятельность людей и т. д. В настоящее время зарегистрирован как международный стандарт ISO/IEC 19501:2005 "Information technology - Open Distributed Processing - Unified Modeling Language (UML)".

UML-модель может включать в себя следующие аспекты;

1. Структурный аспект: Use-Case-диаграммы, идентифицирующие бизнес-процессы и бизнес-транзакции, их взаимосвязь, соподчиненность и взаимодействие; Package-диаграммы, описывающие структуру предметной области и иерархическую структуру организации.

2. Динамический аспект: Behavior-диаграммы (Activity, Statechart, Collaboration, Sequence), описывающие поведение (жизненный цикл) бизнес-процессов в их взаимодействии во времени и в пространстве с привязкой к используемым ресурсам и получаемым результатам.

3. Статический аспект: Class-диаграммы, отражающие совокупность взаимосвязанных объектов, т.е. рассматривающие логическую структуру предметной области, ее внутренние концепции, иерархию объектов и статические связи между ними, структуры данных и объектов; Deployment-диаграммы, отражающие технологические ресурсы организации.

Словарь языка UML включает три вида строительных блоков:

- сущности;
- отношения;
- диаграммы.

Сущности в UML - это абстракции, являющиеся основными элементами модели. Отношения связывают различные сущности; диаграммы группируют представляющие интерес совокупности сущностей.

В UML имеется четыре типа сущностей:

- структурные;
- поведенческие;
- группирующие;
- аннотационные.

Структурные сущности - это имена существительные в моделях на языке UML. Как правило, они представляют собой статические части модели, соответствующие концептуальным или физическим элементам системы. Существует семь разновидностей структурных сущностей: Класс, Интерфейс, Кооперация, Прецедент, Активный класс, Компонент, Узел.

Поведенческие сущности являются динамическими составляющими модели UML. Это глаголы языка: они описывают поведение модели во времени и пространстве. Существует всего два основных типа поведенческих сущностей: Взаимодействие и Автомат.

Группирующие сущности являются организующими частями модели UML. Это блоки, на которые можно разложить модель. Есть только одна первичная группирующая сущность, а именно - пакет.

Аннотационные сущности - пояснительные части модели UML. Это комментарии для дополнительного описания, разъяснения или замечания к любому элементу модели. Имеется только один базовый тип аннотационных элементов - примечание.

Все разновидности сущностей UML в диаграммах имеют свой способ графического изображения.

В языке UML определены четыре типа отношений:

- зависимость;
- ассоциация;
- обобщение;
- реализация.

Диаграмма в UML - это графическое представление набора элементов, изображаемое чаще всего в виде связанного графа с вершинами (сущностями) и ребрами (отношениями). Диаграммы рисуют для визуализации системы с разных точек зрения. Теоретически диаграммы могут содержать любые комбинации сущностей и отношений. На практике, однако, применяется сравнительно небольшое количество типовых комбинаций, соответствующих пяти наиболее употребительным видам, которые составляют архитектуру ИС. Таким образом, в UML выделяют девять типов диаграмм:

- диаграммы классов (Class Diagrams);
- диаграммы объектов (Objects Diagrams);
- диаграммы прецедентов (Use Cases Diagrams);
- диаграммы последовательностей (Sequence Diagrams);
- диаграммы кооперации (Collaboration Diagrams);
- диаграммы состояний (State Diagrams);

- диаграммы действий (Activity Diagrams);
- диаграммы компонентов (Component Diagrams);
- диаграммы развертывания (Deployment Diagram).

Инструментальные средства, поддерживающие методологию UML - Rational Rose (Rational Software), Paradigm Plus (CA/Platinum), ARIS (IDS Sheer AG), Together Designer (Borland) и др..

Система Rational Rose позволяет строить восемь типов диаграмм UML: диаграммы прецедентов, диаграммы классов, диаграммы последовательностей, диаграммы кооперации, диаграммы состояний, диаграммы действий, компонентные диаграммы, диаграммы развертывания. Основным типом диаграмм, своеобразным ядром моделирования в UML являются диаграммы классов. Кроме UML, предусмотрено использование и других методов (Booch, OMT).

Система Paradigm Plus ориентирована на методологию OOCL (Object Oriented Change and Learning) и компонентную технологию проектирования и разработки. Она поддерживает диаграммы различных методов (UML, CLIPP, TeamFusion, OMT, Booch, OOCL, Martin/Odell, Shlaer/Mellor, Coad/Yourdon).

Система ARIS обеспечивает четыре различных "взгляда" на моделирование и анализ: Процессы, Функции (с Целями), Данные, Организация. Для каждого "взгляда" поддерживаются три уровня анализа (требования, спецификации, внедрение). Каждый из уровней анализа состоит из своего комплекта моделей различных типов, в том числе диаграмм UML, диаграмм SAP/R3 и др. Каждый объект моделей ARIS имеет множество атрибутов, которые позволяют контролировать процесс разработки моделей, определять условия для выполнения функционально-стоимостного анализа, имитационного моделирования, взаимодействия с workflow-системами и т.д.

Система Together Designer Community Edition - средство создания диаграмм UML 2.0. Позволяет строить диаграммы прецедентов (диаграммы сценариев взаимодействия пользователя с продуктом с точки зрения пользователя); диаграммы последовательностей, описывающие порядок передачи сообщений от одних объектов к другим; диаграммы кооперации, описывающие взаимодействие объектов друг с другом, диаграммы деятельности, описывающие потоки работ и изменение состояний объектов, диаграммы развертывания. При необходимости может создаваться логическая модель данных, содержащая диаграммы "сущность-связь", на ее основе генерируется физическая модель данных для конкретной СУБД, выбранной для реализации проекта. На этапе создания клиентского и серверного кода все современные средства UML-моделирования могут осуществлять генерацию кода на различных языках программирования.

5.4.1. Моделирование данных с применением UML

Помимо прочего, язык UML применяется для проектирования реляционных БД. Для этого используется небольшая часть языка (диаграммы классов), да и то не в полном объеме. С точки зрения проектирования реляционных БД модельные возможности не слишком отличаются от возможностей ER-диаграмм.

Диаграммой классов в терминологии UML называется диаграмма, на которой показан набор классов (и некоторых других сущностей, не имеющих явного отношения к проектированию БД), а также связей между этими классами. Ограничения могут неформально задаваться на естественном языке или формулироваться на языке объектных ограничений OCL (Object Constraints Language).

Классом называется именованное описание совокупности объектов с общими атрибутами, операциями, связями и семантикой. Графически класс изображается в виде прямоугольника (см. рис. 5.9). Имя (текстовая строка), служит для идентификации класса.

Атрибутом класса называется именованное свойство класса, описывающее множество значений, которые могут принимать экземпляры этого свойства. Класс может иметь любое число атрибутов (в частности, не иметь ни одного атрибута).

Операцией класса называется именованная услуга, которую можно запросить у любого объекта этого класса. Операция - это абстракция того, что можно делать с объектом. Класс может содержать любое число операций (в частности, не содержать ни одной операции). Набор операций класса является общим для всех объектов данного класса.

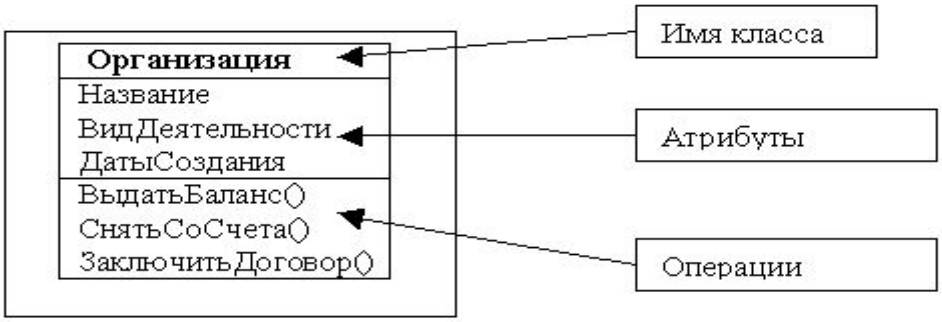

Рис. 5.9. Пример графического представления Класса

В диаграмме классов могут участвовать связи трех разных категорий: зависимость (dependency), обобщение (generalization) и ассоциация (association).

Зависимостью называют связь по применению, когда изменение в спецификации одного класса может повлиять на поведение другого класса, использующего первый класс. Если интерфейс второго класса изменяется, это влияет на поведение объектов первого класса. Зависимость показывается прерывистой линией со стрелкой, направленной к классу, от которого имеется зависимость (см. рис.5.10).

Рис.5.10. Категория связи – зависимость

Связью-обобщением называется связь между общей сущностью, называемой суперклассом, или родителем, и более специализированной разновидностью этой сущности, называемой подклассом, или потомком. Обобщения иногда называют связями "is a", имея в виду, что класс-потомок является частным случаем класса-предка. Класс-потомок наследует все атрибуты и операции класса-предка, но в нем могут быть определены дополнительные атрибуты и операции (см. рис. 5.11).

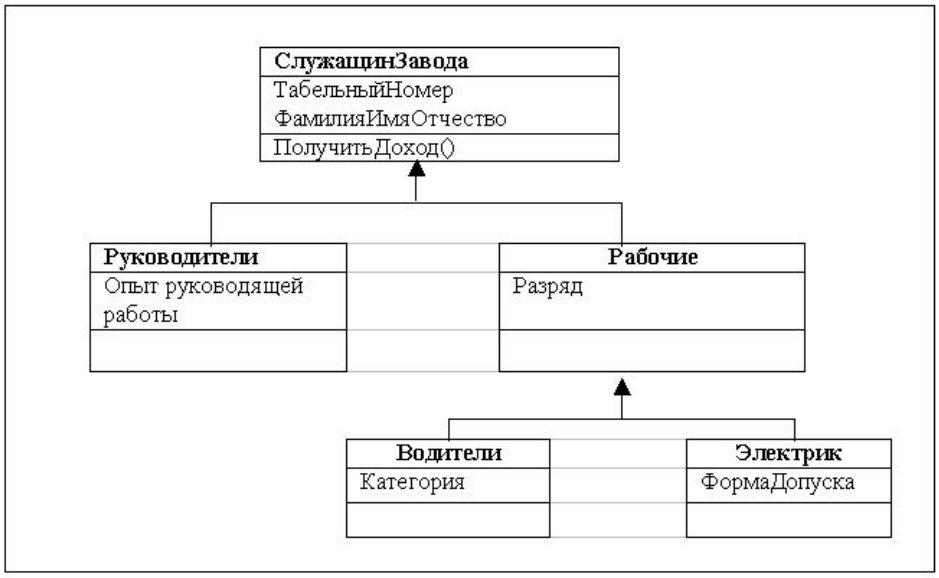

Рис. 5.11. Категория связи – обобщение

Одиночное наследование, когда у каждого подкласса имеется только один суперкласс), является достаточным в большинстве случаев применения связи-обобщения. Однако в UML допускается и множественное наследование, когда один подкласс определяется на основе нескольких суперклассов (см. рис. 5.12).

На этой диаграмме классы Студент и Исследователь порождены из одного суперкласса ЧеловекИзУниверситета. К классу Студент относятся те объекты класса ЧеловекИзУниверситета, которые соответствуют студентам, а к классу Исследователь - объекты класса ЧеловекИзУниверситета, соответствующие исследователям.

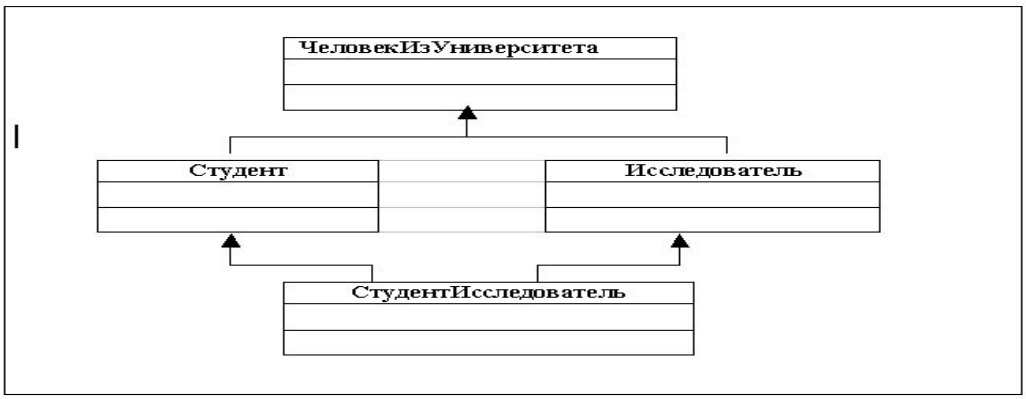

Рис. 5.12. Пример множественного наследования

Часто случается, многие студенты уже в студенческие годы начинают участвовать в исследовательских проектах, так что могут существовать такие два объекта классов Студент и Исследователь, которым соответствует один объект класса ЧеловекИзУниверситета. Таким образом среди объектов класса Студент могут быть исследователи, а некоторые исследователи могут быть студентами. Тогда можно определить класс СтудентИсследователь путем множественного наследования от суперклассов Студент и Исследователь. Объект класса СтудентИсследователь обладает всеми свойствами и операциями классов Студент и Исследователь и может быть использован везде, где могут применяться объекты этих классов. Так что полиморфизм по включению продолжает работать. Однако это влечет за собой ряд проблем. Например, при образовании подклассов Студент и Исследователь в них обоих может быть определен атрибут с именем

Ip_адресКомпьютера. Для объектов класса Студент значениями этого атрибута будут ip-адрес компьютера терминального класса, а для объектов класса Исследователь - ip-адрес компьютера исследовательской лаборатории. При этом для объекта класса СтудентИсследователь могут быть существенны оба одноименных атрибута (у студента-исследователя могут иметься и ip-адрес компьютера терминального класса и ip-адрес компьютера исследовательской лаборатории). На практике применяется одно из следующих решений:

- запретить образование подкласса СтудентИсследователь, пока в одном из суперклассов не будет произведено переименование атрибута Ip_адресКомпьютера;
- наследовать это свойство только от одного из суперклассов, так что, например, значением атрибута Ip_адресКомпьютера у объектов класса СтудентИсследователь всегда будут ip-адресом компьютера исследовательской лаборатории;
- унаследовать в подклассе оба свойства, но автоматически переименовать оба атрибута, чтобы прояснить их смысл; назвать их, например, Ip_адресКомпьютераСтудента и Ip_адресКомпьютераИсследователя.

Каждое из указанных решений имеет недостатки, поэтому при использовании UML для проектирования реляционных БД нужно очень осторожно использовать наследование классов вообще и стараться избегать множественного наследования.

Ассоциацией называется структурная связь, показывающая, что объекты одного класса некоторым образом связаны с объектами другого или того же самого класса. Допускается, чтобы оба конца ассоциации относились к одному классу. В ассоциации могут связываться два класса, и тогда она называется бинарной. Допускается создание ассоциаций, связывающих сразу n классов (они называются n-арными ассоциациями). Графически ассоциация изображается в виде линии, соединяющей класс сам с собой или с другими классами.

С понятием ассоциации связаны четыре важных дополнительных понятия: имя, роль, кратность и агрегация. Ассоциации может быть присвоено имя, характеризующее природу связи. Другим способом именования ассоциации является указание роли каждого класса, участвующего в этой ассоциации. Роль класса, как и имя конца связи в ER-модели, задается именем, помещаемым под линией ассоциации ближе к данному классу.

В общем случае, для ассоциации могут задаваться и ее собственное имя, и имена ролей классов. Связано это с тем, что класс может играть одну и ту же роль в разных ассоциациях, так что в общем случае пара имен ролей классов не идентифицирует ассоциацию. В простых случаях, когда между двумя классами определяется только одна ассоциация, можно вообще не связывать с ней дополнительные имена.

Кратностью (multiplicity) роли ассоциации называется характеристика, указывающая, сколько объектов класса с данной ролью может или должно участвовать в каждом экземпляре ассоциации.

Наиболее распространенным способом задания кратности роли ассоциации является указание конкретного числа или диапазона. Указание "1" говорит о том, что каждый объект класса с данной ролью должен участвовать в некотором экземпляре данной ассоциации, причем в каждом экземпляре ассоциации может участвовать только один объект класса с данной ролью.

Указание диапазона "0..1" говорит о том, что не все объекты класса с данной ролью обязаны участвовать в каком-либо экземпляре данной ассоциации, но в каждом экземпляре ассоциации может участвовать только один объект. Аналогично, указание диапазона "1..*" говорит о том, что все объекты класса с данной ролью должны участвовать в некотором экземпляре данной ассоциации, и в каждом экземпляре ассоциации должен участвовать хотя бы один объект (верхняя граница не задана). В более сложных случаях определения кратности можно использовать списки диапазонов (см. рис. 5.13).

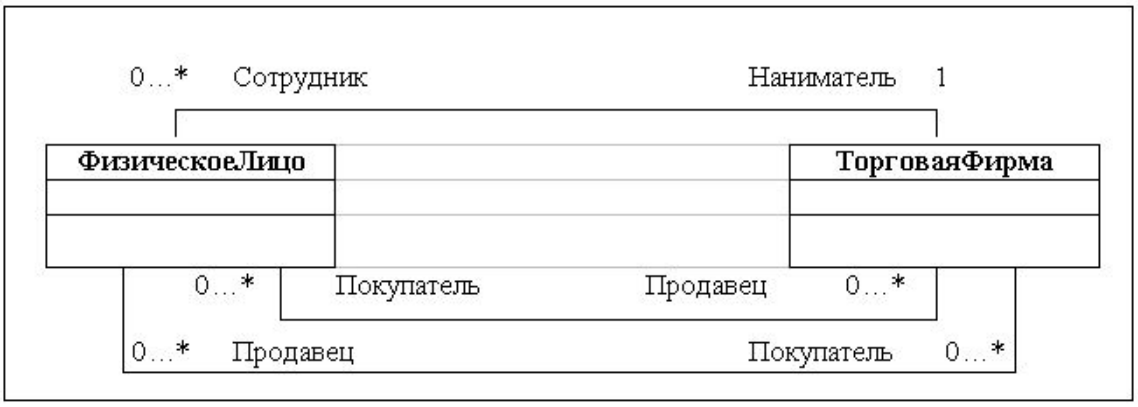

Рис.5.13. Пример объектов класса с определение кратности роли ассоциации

Обычная ассоциация между двумя классами характеризует связь между равноправными сущностями: оба класса находятся на одном концептуальном уровне. Но иногда в диаграмме классов требуется отразить тот факт, что ассоциация между двумя классами имеет специальный вид "часть-целое" (см. рис.5.14).

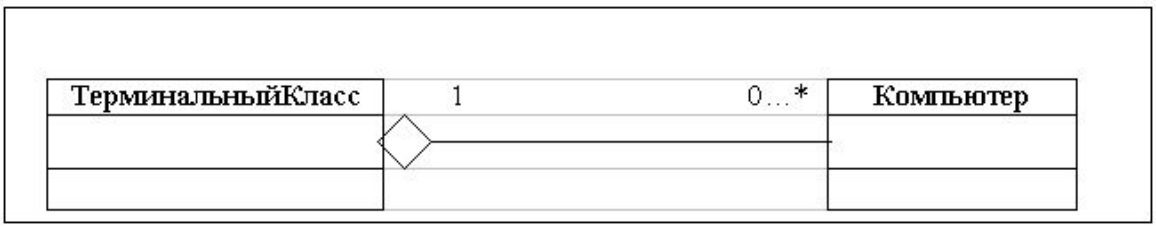

Рис.5.14.Ассоциация вида «часть-целое» - агрегатная

В этом случае класс "целое" имеет более высокий концептуальный уровень, чем класс "часть". Ассоциация такого рода называется агрегатной.

В каждом терминальном классе должны быть установлены компьютеры. Поэтому класс Компьютер является "частью" класса ТерминальныйКласс. Роль класса Компьютер необязательна, поскольку принципиально могут существовать терминальные классы без компьютеров. При этом, хотя терминальные классы, не оснащенные компьютерами, не выполняют своей функции, объекты классов ТерминальныйКласс и Компьютер существуют независимо.

Бывают случаи, когда связь "части" и "целого" настолько сильна, что уничтожение "целого" приводит к уничтожению всех его "частей". Агрегатные ассоциации, обладающие таким свойством, называются композитными, или просто композициями (см. рис. 5.15).

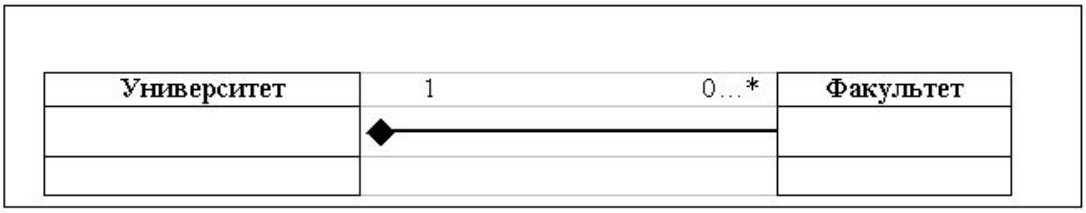

Рис.5.15.Пример агрегатной ассоциации – композитной

Любой факультет является частью одного университета, и ликвидация университета приводит к ликвидации всех существующих в нем факультетов, хотя во время существования университета отдельные факультеты могут ликвидироваться и создаваться.

В диаграммах классов могут указываться ограничения целостности, которые должны поддерживаться в проектируемой БД. В UML допускаются два способа определения ограничений: на естественном языке и на языке OCL (Object Constraints Language).

С точки зрения определения ограничений целостности баз данных более важны средства определения инвариантов классов.

Под инвариантом класса в OCL понимается условие, которому должны удовлетворять все объекты данного класса. Если говорить более точно, инвариант класса - это логическое выражение, вычисление которого должно давать истину при создании любого объекта данного класса и сохранять истинное значение в течение всего времени существования этого объекта. При определении инварианта требуется указать имя класса и выражение, определяющее инвариант указанного класса. Синтаксически это выглядит следующим образом:

```
context <class_name> inv:
<OCL-выражение>
```

Ниже приведены примеры инвариантов для задания ограничений на языке OCL.

1. Выразить на языке OCL ограничение, в соответствии с которым в отделах с номерами больше 5 должны работать сотрудники старше 30 лет.

```
context Отдел inv:
self.номер  5 or
self.служащий  select (возраст 30)  size () = 0
```

2. Определить ограничение, в соответствии с которым у каждого отдела должен быть менеджер и любой отдел должен быть основан не раньше соответствующей компании.

```
context Отдел inv:
self.служащий  exists (должность = “manager”) and
self.компания.годОснования  self.годОснования
```

5.4.2. Перспективы объектного подхода к моделированию данных

Одной из последних разработок в области моделирования предприятия является создание специального унифицированного языка моделирования UEML (Unified Enterprise Modeling Language). Разработка UEML - сетевой проект (IST-2001-34229), финансируемый Евросоюзом.

Проект UEML включает создание:

- общего, визуального, базированного на шаблонах языка для коммерческих инструментальных средств моделирования предприятий и программных систем класса workflow;
- стандартизованных, независимых от инструментов механизмов передачи моделей между проектами;
- репозитория моделей предприятий.

Данный проект осуществляется в соответствии с международными стандартами:

- ISO 14258 Rules and Guidelines for Enterprise Models (Правила и руководящие принципы для моделей предприятия);

- ISO 15704 Requirements for enterprise-reference architectures and methodologies (Требования и методологии по описанию архитектуры предприятия).

В нашей стране в списке действующих ГОСТов по разработке автоматизированных систем (по данным Стандартинформ) следующие:

- ГОСТ 34.003-90 "Информационная технология. Комплекс стандартов на автоматизированные системы. Термины и определения";
- ГОСТ 34.201-89 "Информационная технология. Комплекс стандартов на автоматизированные системы. Виды, комплектность и обозначение документов при создании автоматизированных систем";
- ГОСТ 34.601-90 "Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания";
- ГОСТ 34.602-89 "Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы".

На разработку программной документации действуют стандарты класса ЕСПД (ГОСТ 19.101-77 "Единая система программной документации. Общие положения" и т.д.)

5.5 Сравнение методик

В функциональных моделях (DFD-диаграммах потоков данных, SADT-диаграммах) главными структурными компонентами являются функции (операции, действия, работы), которые на диаграммах связываются между собой потоками объектов.

Несомненным достоинством функциональных моделей является реализация структурного подхода к моделированию объектов данных и связей между ними по принципу «сверху-вниз», когда каждый функциональный блок может быть декомпозирован на множество подфункций и т.д., выполняя, таким образом, модульное проектирование БД. Для функциональных моделей характерны процедурная строгость декомпозиции и наглядность представления.

При функциональном подходе объектные модели данных в виде ER-диаграмм «объект — свойство — связь» разрабатываются отдельно. Для проверки корректности моделирования предметной области между функциональными и объектными моделями устанавливаются взаимно однозначные связи.

Главный недостаток функциональных моделей заключается в том, что процессы и данные существуют отдельно друг от друга — помимо функциональной декомпозиции существует структура данных, находящаяся на втором плане. Кроме того, не ясны условия выполнения процессов обработки информации, которые динамически могут изменяться.

Перечисленные недостатки функциональных моделей снимаются в объектно-ориентированных моделях, в которых главным структурообразующим компонентом выступает класс объектов с набором функций, которые могут обращаться к атрибутам этого класса. Для классов объектов характерна иерархия обобщения, позволяющая осуществлять наследование не только атрибутов (свойств) объектов от вышестоящего класса объектов к нижестоящему классу, но и функций (методов).

В случае наследования функций можно абстрагироваться от конкретной реализации процедур (абстрактные типы данных), которые отличаются для определенных подклассов ситуаций. Это дает возможность обращаться к подобным программным модулям по общим именам (полиморфизм) и осуществлять повторное использование программного кода при модификации программного обеспечения. Таким образом, адаптивность объектно-

ориентированных систем к изменению предметной области по сравнению с функциональным подходом значительно выше.

При объектно-ориентированном подходе изменяется и принцип проектирования. Сначала выделяются классы объектов, а далее в зависимости от возможных состояний объектов (жизненного цикла объектов) определяются методы обработки (функциональные процедуры), что обеспечивает наилучшую реализацию динамического поведения информационной системы.

Для объектно-ориентированного подхода разработаны графические методы моделирования предметной области, обобщенные в языке унифицированного моделирования UML. Однако по наглядности представления модели пользователю-заказчику объектно-ориентированные модели явно уступают функциональным моделям.

При выборе методики моделирования предметной области обычно в качестве критерия выступает степень ее динамичности. Для более регламентированных задач больше подходят функциональные модели, для более адаптивных бизнес-процессов (управления рабочими потоками, реализации динамических запросов к информационным хранилищам) — объектно-ориентированные модели. Однако в рамках одной и той же ИС для различных классов задач могут требоваться различные виды моделей, описывающих одну и ту же проблемную область. В таком случае должны использоваться комбинированные модели предметной области.

ПРАКТИЧЕСКИЕ ЗАДАНИЯ

Практическое задание 5.1

(Выполнить с использованием нотации DFD в BPwin)

1. Постановка задачи «Университетская лаборатория»

Сведения этапа начальной разработки в основном получаются из опросов основных и конечных пользователей. Эти люди являются основными клиентами БД и необходимо идентифицировать с особой тщательностью.

Основными пользователями приложения Университетская Компьютерная Лаборатория (UCL, University Computer Lab) являются:

- заместитель декана (декан);
- директор компьютерной лаборатории (CLD, Computer Lab Director), в чьи обязанности входит оперативное руководство лабораторией;
- ассистенты компьютерной лаборатории (LA, Lab Assistant), в чьи обязанности входит ежедневное обеспечение функционирования лаборатории;
- секретарь компьютерной лаборатории (CLS, Computer Lab Secretary), который помогает выполнять основные административные функции;
- дипломированные ассистенты компьютерной лаборатории (GA, Graduate Assistants), работающие под руководством директора лаборатории и обеспечивающие техническую поддержку и обучение преподавательского состава и сотрудников на оборудовании.

Задачи UCL

Университетская Компьютерная Лаборатория (UCL) расположена в центре территории университета и доступна всем студентам университета независимо от их специализации. Лаборатория UCL предоставляет доступ к многочисленным ресурсам всем сотрудникам университета. В ее распоряжении имеется 200 компьютеров, лазерные принтеры, сканеры. Лаборатория предоставляет сервис и поддержку группам пользователей, составленных из преподавателей, сотрудников или студентов.

В число задач лаборатории входят:

- обеспечение пользователей контролируемым доступом к имуществу UCL, т.е. к микрокомпьютерам, принтерам, материалам, прикладному программному обеспечению и программной документации;
- сопровождение пользователей, работающих на оборудовании UCL, и оперативное решение возникающих проблем. Эти службы первоначально создавались для обучения пользователей основным операциям, выполняемым на компьютере (например, форматирование диска, копирование файлов, установка разрешение программного обеспечения и включение/выключение компьютера).

Описание функций

После определения предназначения UCL и ее организационной структуры можно приступить к изучению ее деятельности. Лаборатория UCL осуществляет шесть видов функций:

- управление инвентаризацией/хранением/заказами;
- поддержка оборудования и организация ремонта;
- выдача и регистрация оборудования;
- ведение платежных ведомостей ассистентов;
- резервирование лаборатории для проведения занятий;
- управление доступом в лабораторию.

2. Выполнение работы.

1. Провести анализ предметной области (см. постановку задачи).
2. Определить внешние объекты, с которыми система должна быть связана –внешние сущности.
3. Выявить основные виды (потоки) информации, циркулирующей между системой и внешними объектами.
4. Построить предварительную контекстную диаграмму, на которой основные функциональные группы представляются процессами, внешние объекты - внешними сущностями, основные виды информации - потоками данных между процессами и внешними сущностями.
5. Провести анализ предварительной контекстной диаграммы и внести в нее необходимые изменения по результатам ответов на возникающие при анализе вопросы по всем ее частям.
6. Построить контекстную диаграмму путем объединения всех процессов предварительной диаграммы в один процесс, а также группирования потоков.
7. Сформировать DFD первого уровня на базе процессов предварительной контекстной диаграммы.
8. Проверить основные требования по DFD первого уровня.
9. Провести декомпозицию каждого процесса текущей DFD с помощью детализирующей диаграммы или спецификации процесса.
10. Проверить основные требования по DFD соответствующего уровня.
11. По необходимости добавить определение новых потоков в словарь данных при каждом их появлении на диаграммах.
12. Параллельно с процессом декомпозиции изучить требования (в том числе и вновь поступающие), провести их разбиение на элементарные и идентифицировать процессы.
13. После двух-трех уровней декомпозиции провести ревизию с целью проверки корректности и улучшения понимаемости модели.

Практическое задание 5.2

(Выполнить с использованием нотации IDEF1X в ERwin)

Проектирование модели БД на примере пункта обмена валют.

Этапы разработки модели БД.

1. Постановка задачи.
2. Создание логической модели БД.
3. Создание физической модели БД.
4. Генерация схемы БД.

Создание логической модели БД включает в себя:

- Анализ предметной области, выделение основных сущностей.
- Определение связей между сущностями.
- Указание первичных ключей и неключевых атрибутов.

1. Постановка задачи. В гипотетическом пункте обмена валют создается локальная информационная система (ИС), призванная автоматизировать процесс учета сделок купли-продажи валюты. Создаваемая система должна обеспечить ввод, хранение и поиск информации о сделках, совершенных в данном пункте обмена валют. Каждой сделке присваивается уникальный цифровой код. Информация о сделке должна содержать сведения о дате, времени сделки, суммах покупаемой о продаваемой валют, фамилии, имени, отчестве и номере паспорта клиента, а также о фамилии, инициалах и учетном номере личного дела кассира в отделе кадров. Система должна позволять вычислить денежный оборот за один или несколько дней, а также осуществлять поиск информации о сделках по номеру паспорта клиента. Задача состоит в проектировании структуры базы данных разрабатываемой автоматизированной ИС.

2. Создание логической модели БД

Проведите анализ предметной области с целью выделить основные сущности.

Определение связей между сущностями

Для задания связей между сущностями сначала составьте описание данной предметной области при помощи ряда истинных высказываний на естественном языке.

На основе анализа приведенных высказываний выделите связи (названия связей - глаголы), определите тип, мощность.

Укажите первичные ключи и неключевые атрибуты

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

1. Дайте характеристику понятия моделирование данных.
2. Каковы задачи методологии структурного анализа данных?
3. Каково назначение методологии диаграмм потоков данных?
4. Что такое поток данных в методологии DFD?
5. Какова функция хранилища данных в DFD?
6. Что такое миниспецификация в DFD - модели? Каким требованиям должны удовлетворять миниспецификации?
7. Каким образом диаграмма потоков данных должна быть проверена на полноту и непротиворечивость?
8. Преимущества и недостатки методики DFD.
9. В чем сходство и в чем различие методологии структурного анализа данных и диаграмм потоков данных?
10. Что из себя представляют концепция и семантика методики IDEF1X?
11. Каким образом производится идентификация сущностей в IDEF1X?
12. Какие вы знаете инструменты поддержки стандартов моделирования?
13. Каким требованиям должен удовлетворять современный инструмент моделирования баз данных?
14. Дайте характеристику AllFusion ERwin Data Modeler.
15. Дайте характеристику объектному подходу к моделированию данных с использованием UML.
16. Какие типы сущностей имеются в UML?
17. Какие типы отношений определены в языке UML?
18. Какие типы диаграмм выделяют в языке UML?
19. Какие вам известны инструментальные средства, поддерживающие методологию UML?
20. Что такое диаграмма классов в терминологии UML?
21. Какие категории связи участвуют в диаграмме классов UML?
22. Какие вам известны действующие ГОСТы по разработке автоматизированных систем?

ГЛАВА 6. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

Проектирование БД – одна из наиболее сложных и ответственных задач, связанных с созданием информационной системы. В результате решения этой задачи должны быть определены содержание БД, эффективный для всех её будущих пользователей способ организации данных и инструментальные средства управления данными. В крупных системах проектирование БД требует особой тщательности, поскольку цена допущенных на этой стадии просчётов и ошибок особенно велика. Некоторые ошибки проектирования можно скорректировать позже в процессе эксплуатации с помощью средств реструктуризации и реорганизации БД, но такие операции являются весьма трудоемкими и дорогостоящими.

6.1. Теория проектирования баз данных

Главной задачей проектирования базы данных является определение состава и структуры базы данных, способа ее организации, выбор инструментальных средств управления данными. Процесс проектирования включает в себя следующие этапы:

1. Концептуальное проектирование.
2. Логическое проектирование.
3. Физическое проектирование.
4. Опытная эксплуатация.

Современные CASE-системы позволяют сократить число этапов при проектировании системы. После создания концептуальной модели они автоматически выполняют физическое проектирование БД для выбранной СУБД.

Основная цель процесса проектирования БД состоит в получении такого проекта, который удовлетворяет следующим требованиям:

1. Корректность схемы БД, т.е. база должна быть гомоморфным образом моделируемой предметной области (далее – ПО), где каждому объекту ПО соответствуют данные в памяти ЭВМ, а каждому процессу – адекватные процедуры обработки данных.
2. Обеспечение ограничений (на объёмы внешней и оперативной памяти и другие ресурсы вычислительной системы).
3. Эффективность функционирования (соблюдение ограничений на время реакции системы на запрос и обновление данных).
4. Защита данных (от сбоев и несанкционированного доступа).
5. Простота и удобство эксплуатации.
6. Гибкость, т.е. возможность развития и адаптации к изменениям ПО и/или требований пользователей.

Этап создания концептуальной модели предполагает выделение объектов предметной области, подлежащих описанию, и установление логических связей между этими объектами и заканчивается описанием модели предметной области в виде некоторой структуры данных, соответствующей объектам предметной области, и установлением связи между данными. Созданную концептуальную модель данных во многих источниках называют еще инфологической моделью предметной области.

Концептуальное проектирование состоит из следующих этапов:

1. Идентификация сущностей (выделение объектов рассматриваемой предметной области).

2. Установление всех (структурных, иерархических, запросных) связей между сущностями.

3. Определение атрибутов сущностей (существенных свойств объектов).

4. Нормализация модели.

5. Минимизация числа сущностей.

Этап логического проектирования предназначен для создания на основе концептуальной модели предметной области логической модели будущей БД в среде конкретной СУБД (данные и логические связи между ними). При этом если к моменту логического проектирования СУБД не задано, то этому этапу должен предшествовать этап выбора СУБД. Логическая модель может быть иерархической, сетевой или реляционной. Для реляционных СУБД логическое проектирование заканчивается созданием реляционной модели (схемы) БД, включающей полный список отношений и их атрибутов с указанием первичного ключа.

На этапе концептуального проектирования данные рассматриваются без учета специфики используемой СУБД. Для выбора инструментального средства реализации концептуальной модели предметной области и создания соответствующей этому выбору СУБД-ориентированной схемы базы данных производится этап логического (его еще называют даталогическим) проектирования. Как уже говорилось ранее, современные инструментальные средства СУБД позволяют реализовать не только логическую модель данных, но и создать физическую модель, то есть совместить два этапа – этап логического и физического проектирования.

Основной задачей этапа логического проектирования является выбор СУБД и разработка СУБД-ориентированной схемы, которая удовлетворяет всему диапазону требований пользователей, начиная с требований целостности и непротиворечивости проектируемой базы данных и заканчивая показателями эффективности функционирования при ее расширении и усложнении. При этом одновременно проводится работа по составлению прикладных программ.

Прежде чем приступать к логическому проектированию, разработчик должен иметь следующие данные:

1. Концептуальную модель предметной области (СУБД-независимую схему).

2. Характеристики одной или нескольких СУБД.

3. Вычислительные средства – ограничения на конфигурацию и объем аппаратного и программного обеспечения.

4. Руководство для группы сопровождения БД – администратора и обслуживающего персонала.

Процесс логического проектирования состоит из следующих шагов после выбора инструментального средства:

1. Определение локальных информационных структур. Вначале анализируются требования к обработке данных. Затем на основании этих требований создаются отдельные схемы локальных информационных структур. При этом формат локальных информационных структур, подлежащих обработке, соответствует формату отдельных сущностей и сущностей-связей, полученных в процессе концептуального проектирования.

2. Формирование первоначального варианта СУБД-ориентированной схемы. После того, как составлены конкретные СУБД-схемы для отдельных сущностей, они объединяются в новую информационную структуру. При этом в простейшем случае сущностям будут соответствовать файлы определенных типов записей, а атрибутам – поля записей. В более сложных случаях возможные сущности могут быть расщеплены на несколько составляющих или объеденены в одну. На этом этапе, безусловно, учитываются

особенности выбранной СУБД. На этом же этапе проектируются прикладные программы обработки данных.

3. Оценка характеристик полученной СУБД-схемы. Построенная на втором этапе логическая структура базы данных может быть оценена количественно с помощью таких характеристик, как число обращений к логическим записям, объем обрабатываемых в каждом приложении данных, общий объем хранимых данных. Эти оценки помогут определить эффективность функционирования БД.

Следующий этап предназначен для создания физической модели БД, которая состоит из описания всех типов файлов БД. Он заканчивается генерацией БД для заданной СУБД. В современных CASE - системах физическая модель БД для заданной СУБД создается автоматически на основе концептуальной модели предметной области. При этом физическая модель, созданная для одной СУБД, может быть автоматически инвертирована в другую из списка, поддерживаемого CASE-системой.

Этап опытной эксплуатации предназначен для проверки работоспособности БД и ее эффективности. Ответственной за опытную эксплуатацию является администрация БД. Опытная эксплуатация проводится на технике заказчика при непосредственном его участии. Срок опытной эксплуатации зависит от спецификации проекта.

6.1.1. Концептуальное проектирование

Концептуальный подход не предоставляет формальных способов моделирования реальности, однако он закладывает основы методологии проектирования БД. Первой задачей концептуального проектирования является определение предметной области (ПО) системы, позволяющее изучить информационные потребности будущих пользователей. Другая задача этого этапа – анализ ПО, который призван сформировать взгляд на ПО с позиций сообщества будущих пользователей БД. Анализ ПО выполняется разработчиком логической базы данных – специалистом в данной предметной области.

Концептуальная модель предметной области (ПО) представляет собой описание структуры и динамики ПО, характера информационных потребностей пользователей системы в терминах, понятных пользователю и независимых от реализации системы. Более того, концептуальная модель ПО не должна зависеть от модели данных, которая будет использована при создании БД. Обычно описание ПО выражается в терминах не отдельных объектов и связей между ними, а их типов, связанных с ними ограничений целостности и тех процессов ПО, которые приводят к переходу ПО из одного состояния в другое. Такое описание может быть представлено любым способом, допускающим однозначную интерпретацию. В простых случаях описание ПО представляется на естественном языке, в более сложных используется также математический аппарат: таблицы, диаграммы, графы и т.п. Если анализ ПО выполняется несколькими специалистами, то они должны принять соглашения, касающиеся:

- используемых методов анализа предметной области;
- правил именования и обозначения объектов ПО, атрибутов и связей;
- содержания и формата создаваемых ими документов.

Существуют разные подходы к концептуальному проектированию:

1. Функциональный подход к проектированию БД - этот метод является наиболее распространённым; он реализует принцип "от задач" и применяется в том случае, когда известны функции некоторой группы лиц и/или комплекса задач, для обслуживания информационных потребностей которых создаётся рассматриваемая БД;

2. Предметный подход к проектированию БД - этот метод применяется в тех случаях, когда у разработчиков есть чёткое представление о самой ПО и о том, какую именно

информацию они хотели бы хранить в БД, а структура запросов не определена или определена не полностью; тогда основное внимание уделяется исследованию ПО и наиболее адекватному её отображению в БД с учётом самого широкого спектра информационных запросов к ней;

3. Проектирование с использованием метода "сущность–связь"- этот метод (Entity–Relation, ER–method) был разработан в 1976 г. П.Ченом (Chen P.P.), он является комбинацией двух предыдущих и обладает достоинствами обоих.

Последний подход рассмотрим подробнее. Этап концептуального проектирования начинается с моделирования ПО. Проектировщик разбивает ПО на ряд локальных областей, каждая из которых (в идеале) включает в себя информацию, достаточную для обеспечения информационных потребностей одной группы будущих пользователей или решения отдельной задачи. Каждое локальное представление моделируется отдельно, а затем выполняется их объединение. Выбор локального представления зависит от масштабов ПО. Обычно ПО разбивается на локальные области так, чтобы каждая из них соответствовала отдельному внешнему приложению и содержала 6-7 сущностей (т.е. объектов, о которых в системе будет накапливаться информация).

Сущности, существование которых не зависит от существования других сущностей, называются базовыми, остальные сущности – зависимыми. Например, сущность ЛЕКЦИЯ зависит от базовых сущностей ГРУППА, ПРЕПОДАВАТЕЛЬ, ДИСЦИПЛИНА.

Для каждой сущности определяются атрибуты, которые делятся на два типа: идентифицирующие и описательные. Идентифицирующие атрибуты входят в состав ключа (или ключей) и позволяют однозначно распознавать экземпляры сущности. Первичный ключ базовой сущности не может содержать неопределённые значения атрибутов (null). Первичный ключ должен включать в свой состав минимально необходимое для идентификации количество атрибутов. Описательные атрибуты заключают в себе свойства сущности, интересующие пользователей. Спецификация атрибута состоит из его названия, указания типа данных и описания ограничений целостности – множества значений, которые может принимать данный атрибут.

Далее осуществляется спецификация связей: выявляются связи между сущностями внутри локального представления. Каждая связь именуется. При объединении проектировщик может формировать конструкции, производные по отношению к тем, которые были использованы в локальных представлениях. Цель введения подобных абстракций:

- объединение в единое целое фрагментарных представлений о различных свойствах одного и того же объекта;
- введение абстрактных понятий, удобных для решения задач системы, установление их связи с более конкретными понятиями модели;
- образование классов и подклассов подобных объектов (например, класс "изделие" и подклассы типов изделий, производимых на предприятии).

При небольшом количестве локальных областей (не более пяти) объединение выполняется за один шаг. В противном случае обычно выполняют бинарное объединение. При объединении представлений используют три основополагающие концепции:

1. Идентичность. Два или более элементов модели идентичны, если они имеют одинаковое семантическое значение.

2. Агрегация. Позволяет рассматривать связь между элементами как новый элемент. Например, связь экзамен между сущностями СТУДЕНТ, ДИСЦИПЛИНА, ПРЕПОДАВАТЕЛЬ может быть представлена агрегированной сущностью ЭКЗАМЕН с атрибутами Название дисциплины, Фамилия преподавателя, Фамилия студента, Оценка.

3. Обобщение. Позволяет образовывать многоуровневую иерархию обобщений. Например, в объединяемых представлениях присутствуют следующие сущности:

ДЕТАЛИ СОБСТВЕННОГО ПРОИЗВОДСТВА
ДЕТАЛИ ПОКУПНЫЕ
СБОРОЧНЫЕ ЕДИНИЦЫ ПОКУПНЫЕ
СБОРОЧНЫЕ ЕДИНИЦЫ СОБСТВЕННОГО ПРОИЗВОДСТВА
Их можно объединить так, как показано на рис. 6.1.

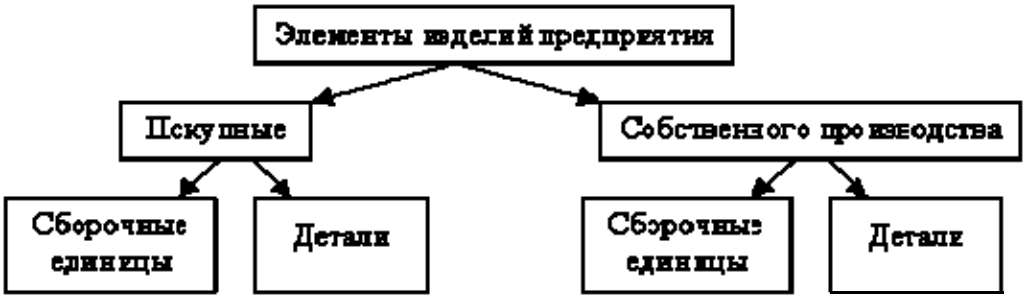

Рис.6.1. Использование обобщений при объединении

Это позволит упростить формализацию процессов обработки данных. Например, оформление заказа на покупные элементы изделий в данном примере может быть описано один раз (для второго уровня иерархии). На этапе объединения необходимо выявить и устранить все противоречия. Например, одинаковые названия семантически различных объектов или связей или несогласованные ограничения целостности на одни и те же атрибуты в разных приложениях. Устранение противоречий вызывает необходимость возврата к этапу моделирования локальных представлений с целью внесения в них соответствующих изменений. По завершении объединения результаты проектирования представляют собой концептуальную инфологическую модель ПО. Модели локальных представлений – это внешние инфологические модели.

На этапе анализа ПО также решаются следующие задачи:

1. Определение правил (ограничений целостности), которым должны удовлетворять сущности ПО, атрибуты сущностей и связи между ними. Часть этих правил реализуется в схеме базы данных (возможности реализации ограничений целостности в схеме БД определяются моделью данных той СУБД, которая будет выбрана для реализации проекта). Остальные правила реализуются с помощью программного обеспечения.
2. Выделение групп пользователей системы. Каждая группа выполняет определённые задачи и обладает разными правами доступа к системе.
3. Создание внешней спецификации тех функций (процессов), которые эта система будет выполнять. Например, для той же библиотечной системы это задачи поиска книг (по определённым критериям), выдачи/приёма книг, определение списка должников и т.д.

6.1.2. Логическое проектирование БД

Определение требований к операционной обстановке. На этом этапе производится оценка требований к вычислительным ресурсам, необходимым для функционирования системы, выбор типа и конфигурации ЭВМ, типа и версии операционной системы.

Выбор зависит от следующих показателей:

- примерный объём данных в БД;
- динамика роста объёма данных;
- характер запросов к данным (извлечение и обновление отдельных записей, групп записей, обработка отдельных отношений или соединение отношений);

- интенсивность запросов к данным по типам запросов;
- требования к времени отклика системы по типам запросов.

Выбор СУБД и инструментальных программных средств. Выбор СУБД является одним из важнейших моментов в разработке проекта БД, так как он принципиальным образом влияет на весь процесс проектирования БД и реализации информационной системы. Теоретически при осуществлении этого выбора нужно принимать во внимание десятки факторов. Но на практике разработчики руководствуются лишь собственной интуицией и несколькими наиболее важными критериями, к которым, в частности, относятся:

- тип модели данных, которую поддерживает данная СУБД, адекватность модели данных структуре рассматриваемой ПО;
- характеристики производительности СУБД;
- запас функциональных возможностей для дальнейшего развития информационной системы;
- степень оснащенности СУБД инструментарием для персонала администрирования данными;
- удобство и надежность СУБД в эксплуатации;
- стоимость СУБД и дополнительного программного обеспечения.

Создание модели данных, поддерживаемой выбранной СУБД. На этапе логического проектирования разрабатывается логическая структура БД, соответствующая концептуальной модели ПО. Решение этой задачи существенно зависит от модели данных, поддерживаемой выбранной СУБД. Результатом выполнения этого этапа являются схемы базы данных концептуального и внешнего уровней архитектуры, составленные на языках определения данных (DDL) выбранной СУБД.

При проектировании базы данных решаются две основных проблемы.

- Каким образом отобразить объекты предметной области в абстрактные объекты модели данных, чтобы это отображение не противоречило семантике предметной области и было, по возможности, лучшим (эффективным, удобным и т.д.). Часто эту проблему называют проблемой логического проектирования баз данных.
- Как обеспечить эффективность выполнения запросов к базе данных, т.е. каким образом, имея в виду особенности конкретной СУБД, расположить данные во внешней памяти, создание каких дополнительных структур (например, индексов) потребовать и т.д.? Эту проблему называют проблемой физического проектирования баз данных.

6.1.3. Физическое проектирование

В случае реляционных баз данных трудно представить какие-либо общие рецепты по части физического проектирования. Здесь слишком много зависит от используемой СУБД. Например, при работе с СУБД Ingres можно выбирать один из предлагаемых способов физической организации отношений, при работе с System R следовало бы, прежде всего, подумать о кластеризации отношений и требуемом наборе индексов и т.д. Поэтому мы ограничимся вопросами логического проектирования реляционных баз данных, которые существенны при использовании любой реляционной СУБД. Более того, мы не будем касаться очень важного аспекта проектирования - определения ограничений целостности (за исключением ограничения первичного ключа). Дело в том, что при использовании СУБД с развитыми механизмами ограничений целостности (например, SQL-ориентированных систем) трудно предложить какой-либо общий подход к определению ограничений целостности. Эти ограничения могут иметь очень общий вид, и их формулировка пока относится скорее к области искусства, чем инженерного мастерства. Самое большее, что

предлагается по этому поводу в литературе, это автоматическая проверка непротиворечивости набора ограничений целостности. Так что будем считать, что проблема проектирования реляционной базы данных состоит в обоснованном принятии решений о том, из каких отношений должна состоять БД и какие атрибуты должны быть у этих отношений.

Этап физического проектирования заключается в увязке логической структуры БД и физической среды хранения с целью наиболее эффективного размещения данных, т.е. отображении логической структуры БД в структуру хранения. Решается вопрос размещения хранимых данных в пространстве памяти, выбора эффективных методов доступа к различным компонентам "физической" БД. Результаты этого этапа документируются в форме схемы хранения на языке определения хранимых данных. Принятые на этом этапе решения оказывают определяющее влияние на производительность системы.

Фактически проектирование БД имеет итерационный характер. В процессе функционирования системы становится возможным изменение её реальных характеристик, выявление "узких" мест. И если система не отвечает предъявляемым к ней требованиям, то обычно она подвергается реорганизации, т.е. модификации первоначально созданного проекта.

Этап опытной эксплуатации предназначен для проверки работоспособности БД и ее эффективности. Ответственной за опытную эксплуатацию является администрация БД. Опытная эксплуатация проводится на технике заказчика при непосредственном его участии. Срок опытной эксплуатации зависит от спецификации проекта.

6.2. Классический подход к проектированию баз данных

Проектирование реляционной базы данных в терминах отношений часто представляет собой очень сложный и неудобный для проектировщика процесс. При этом проявляется ограниченность реляционной модели данных в следующих аспектах.

- Модель не предоставляет достаточных средств для представления смысла данных. Семантика реальной предметной области должна независимым от модели способом представляться в голове проектировщика. В частности, это относится к упоминавшейся нами проблеме представления ограничений целостности.
- Для многих приложений трудно моделировать предметную область на основе плоских таблиц. В ряде случаев на самой начальной стадии проектирования проектировщику приходится производить насилие над собой, чтобы описать предметную область в виде одной (возможно, даже ненормализованной) таблицы.
- Хотя весь процесс проектирования происходит на основе учета зависимостей, реляционная модель не предоставляет каких-либо средств для представления этих зависимостей.
- Несмотря на то, что процесс проектирования начинается с выделения некоторых существенных для приложения объектов предметной области ("сущностей") и выявления связей между этими сущностями, реляционная модель данных не предлагает какого-либо аппарата для разделения сущностей и связей.

Рассмотрим классический подход, при котором весь процесс проектирования производится в терминах реляционной модели данных методом последовательных приближений к удовлетворительному набору схем отношений. Исходной точкой является представление предметной области в виде одного или нескольких отношений, и на каждом шаге проектирования производится некоторый набор схем отношений, обладающих лучшими свойствами. Процесс проектирования представляет собой процесс нормализации

схем отношений, причем каждая следующая нормальная форма обладает свойствами лучшими, чем предыдущая.

6.2.1. Нормализация отношений

Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений. Примером набора ограничений является ограничение первой нормальной формы - значения всех атрибутов отношения атомарны. Поскольку требование первой нормальной формы является базовым требованием классической реляционной модели данных, мы будем считать, что исходный набор отношений уже соответствует этому требованию. В теории реляционных баз данных обычно выделяется следующая последовательность нормальных форм:

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

Основные свойства нормальных форм:

- каждая следующая нормальная форма в некотором смысле лучше предыдущей;
- при переходе к следующей нормальной форме свойства предыдущих нормальных форм сохраняются.

В основе процесса проектирования лежит метод нормализации: декомпозиция отношения, находящегося в предыдущей нормальной форме, в два или более отношения, удовлетворяющих требованиям следующей нормальной формы.

6.2.2. Нормальные формы ER-схем

Как и в реляционных схемах баз данных, в ER-схемах вводится понятие нормальных форм, причем их смысл очень близко соответствует смыслу реляционных нормальных форм. Заметим, что формулировки нормальных форм ER-схем делают более понятным смысл нормализации реляционных схем. Мы приведем только очень краткие и неформальные определения трех первых нормальных форм.

В первой нормальной форме ER-схемы устраняются повторяющиеся атрибуты или группы атрибутов, т.е. производится выявление неявных сущностей, "замаскированных" под атрибуты.

Во второй нормальной форме устраняются атрибуты, зависящие только от части уникального идентификатора. Эта часть уникального идентификатора определяет отдельную сущность.

В третьей нормальной форме устраняются атрибуты, зависящие от атрибутов, не входящих в уникальный идентификатор. Эти атрибуты являются основой отдельной сущности.

Более сложные элементы ER-модели. Мы остановились только на самых основных и наиболее очевидных понятиях ER-модели данных. К числу более сложных элементов модели относятся следующие.

Подтипы и супертипы сущностей. Как в языках программирования с развитыми типовыми системами (например, в языках объектно-ориентированного программирования), вводится возможность наследования типа сущности, исходя из одного или нескольких супертипов. Интересные нюансы связаны с необходимостью графического изображения этого механизма.

Связи "many-to-many". Иногда бывает необходимо связывать сущности таким образом, что с обоих концов связи могут присутствовать несколько экземпляров сущности (например, все члены кооператива сообща владеют имуществом кооператива). Для этого вводится разновидность связи "многие-со-многими".

Уточняемые степени связи. Иногда бывает полезно определить возможное количество экземпляров сущности, участвующих в данной связи (например, служащему разрешается участвовать не более, чем в трех проектах одновременно). Для выражения этого семантического ограничения разрешается указывать на конце связи ее максимальную или обязательную степень.

Каскадные удаления экземпляров сущностей. Некоторые связи бывают настолько сильными (конечно, в случае связи "один-ко-многим"), что при удалении опорного экземпляра сущности (соответствующего концу связи "один") нужно удалить и все экземпляры сущности, соответствующие концу связи "многие". Соответствующее требование "каскадного удаления" можно сформулировать при определении сущности.

Домены. Как и в случае реляционной модели данных бывает полезна возможность определения потенциально допустимого множества значений атрибута сущности (домена).

Эти и другие более сложные элементы модели данных "сущность-связи" делают ее существенно более мощной, но одновременно несколько усложняют ее использование. Конечно, при реальном использовании ER-диаграмм для проектирования баз данных необходимо ознакомиться со всеми возможностями.

6.2.3 Получение реляционной схемы из ER-схемы

Шаг 1. Каждая простая сущность превращается в таблицу. Простая сущность - сущность, не являющаяся подтипом и не имеющая подтипов. Имя сущности становится именем таблицы.

Шаг 2. Каждый атрибут становится возможным столбцом с тем же именем; может выбираться более точный формат. Столбцы, соответствующие необязательным атрибутам, могут содержать неопределенные значения; столбцы, соответствующие обязательным атрибутам, - не могут.

Шаг 3. Компоненты уникального идентификатора сущности превращаются в первичный ключ таблицы. Если имеется несколько возможных уникальных идентификаторов, выбирается наиболее используемый. Если в состав уникального идентификатора входят связи, к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (этот процесс может продолжаться рекурсивно). Для именования этих столбцов используются имена концов связей и/или имена сущностей.

Шаг 4. Связи многие-к-одному (и один-к-одному) становятся внешними ключами. Т.е. делается копия уникального идентификатора с конца связи "один", и соответствующие столбцы составляют внешний ключ. Необязательные связи соответствуют столбцам,

допускающим неопределенные значения; обязательные связи - столбцам, не допускающим неопределенные значения.

Шаг 5. Индексы создаются для первичного ключа (уникальный индекс), внешних ключей и тех атрибутов, на которых предполагается в основном базировать запросы.

Шаг 6. Если в концептуальной схеме присутствовали подтипы, то возможны два способа (см. табл.6.1):

- все подтипы в одной таблице,
- для каждого подтипа - отдельная таблица.

Таблица 6.1

Преимущества и недостатки использования подтипов

Все в одной таблице	Таблица - на подтип
Преимущества	
1.Все хранится вместе. 2.Легкий доступ к супертипу и подтипам. 3.Требуется меньше таблиц.	1.Более ясны правила подтипов. 2.Программы работают только с нужными таблицами.
Недостатки	
1.Слишком общее решение. Требуется дополнительная логика работы с разными наборами столбцов и разными ограничениями. 2.Потенциальное узкое место (в связи с блокировками). 3.Столбцы подтипов должны быть необязательными. 4.В некоторых СУБД для хранения неопределенных значений требуется дополнительная память.	1.Слишком много таблиц. 2.Смущающие столбцы в представлении UNION. 3.Потенциальная потеря производительности при работе через UNION. 4.Над супертипом невозможны модификации.

При применении способа (1) таблица создается для наиболее внешнего супертипа, а для подтипов могут создаваться представления. В таблицу добавляется, по крайней мере, один столбец, содержащий код ТИПА; он становится частью первичного ключа.

При использовании метода (2) для каждого подтипа первого уровня (для более нижних - представления) супертип воссоздается с помощью представления UNION (из всех таблиц подтипов выбираются общие столбцы - столбцы супертипа).

Шаг 7. Имеется два способа работы при наличии исключающих связей:

1. общий домен,
2. явные внешние ключи.

Если остающиеся внешние ключи все в одном домене, т.е. имеют общий формат (способ 1), то создаются два столбца: идентификатор связи и идентификатор сущности. Столбец идентификатора связи используется для различения связей, покрываемых дугой исключения. Столбец идентификатора сущности используется для хранения значений уникального идентификатора сущности на дальнем конце соответствующей связи.

Если результирующие внешние ключи не относятся к одному домену, то для каждой связи, покрываемой дугой исключения, создаются явные столбцы внешних ключей; все эти столбцы могут содержать неопределенные значения (см. табл. 6.2).

Талица 6.2

Работа при наличии исключающих связей

Общий домен	Явные внешние ключи
Преимущества	
Нужно только два столбца.	Условия соединения - явные.
Недостатки	
Оба дополнительных атрибута должны использоваться в соединениях.	Слишком много столбцов.

6.2.4. Автоматизация проектирования БД

Функциональное ядро систем автоматизированного проектирования (САПР) БД строится как совокупность взаимосвязанных модулей инфологического моделирования, проектирования схемы, подсхем и физической организации БД. Существующие в настоящее время САПР БД строятся как человеко-машинные экспертные системы. В первую очередь это определяется слабо - поддающимся формализации процессом синтеза инфологического описания ПО, т.е. преобразования неформальных представлений реального мира в формальные категории. Этот процесс выполняется экспертом – специалистом в той или иной ПО. Поэтому все проблемы, которые характерны для формирования базы знаний экспертной системы, возникают и в случае САПР БД.

Характерной особенностью САПР БД является её ориентация на коллективное творчество и продолжительность самого процесса проектирования, предполагающего множественные итерации. Это находит свое отражение в наличии журнала проектирования и других средств, обеспечивающих ведение и коллективное использование исходных данных, промежуточных и окончательных результатов проектирования.

Как правило, современные средства проектирования данных поддерживают несколько типов СУБД (например, ERwin фирмы Computer Associates поддерживает более 20 различных СУБД). Уровень поддержки той или иной платформы в разных средствах проектирования данных может быть различен. Например, конкретное средство может поддерживать или не поддерживать для данной СУБД такие особенности, как создание хранимых процедур, генерация объектов физической памяти (табличных пространств, сегментов отката и др.), задание местоположения объектов базы данных в физических объектах и т.д. Поэтому, выбирая средство проектирования данных для решения конкретной задачи, стоит поинтересоваться, каковы его возможности с точки зрения поддержки особенностей той или иной платформы - при удачном раскладе можно сэкономить немало времени на «ручное» доведение создаваемой базы данных (или DDL-скрипта для ее генерации) до необходимого состояния. При этом, естественно, чем больше возможностей и платформ поддерживает конкретное средство проектирования данных, тем дороже оно стоит (следует отметить, что CASE-средства вообще относятся к не самым дешевым программным продуктам - цены на них составляют обычно от одной до нескольких десятков тысяч долларов).

Некоторые средства проектирования данных позволяют хранить их модели в специальных репозитариях, а также осуществлять коллективное проектирование. Нередко средства проектирования данных дают возможность также присваивать таблицам и полям так называемые расширенные атрибуты, то есть свойства, связанные с отображением их в

клиентских приложениях, созданных с помощью какого-либо средства разработки, а также генерировать код приложений для ввода и отображения данных.

6.2.5. Наиболее популярные средства проектирования данных

Кратко рассмотрим особенности наиболее популярных CASE-средств, применяемых для проектирования данных (см. табл. 6.3).

Таблица 6.3

Наиболее популярные CASE-средства, применяемые для проектирования данных

CASE – средство	Производитель	URL
Designer 2000	Oracle	http://www.oracle.com/
ERwin	Computer Associates	http://www.cai.com/
PowerDesigner	Sybase	http://www.sybase.com/
ER/Studio	Embarcadero	http://www.embarcadero.com/
System Architect	Popkin Software	http://www.popkin.com/
Visible Analyst	Visible Systems	http://www.visible.com
Visio Enterprise	Microsoft	http://www.microsoft.com/

Многие из этих продуктов предназначены не только для проектирования данных, но и для решения других задач, например моделирования потоков данных или бизнес-процессов, функционального моделирования, прототипирования приложений, их документирования, управления проектами и т.д. В этом случае средства проектирования данных являются составными частями таких продуктов.

Designer/2000 (Oracle). Designer/2000 (предыдущие версии продукта назывались Oracle*CASE) представляет собой универсальное CASE-средство, позволяющее моделировать бизнес-процессы, создавать диаграммы потоков данных и функциональные модели. Средство проектирования данных и создания ER-диаграмм является лишь одной из составных частей этого довольно сложного продукта и предоставляет возможность сохранять созданные модели данных и описанные бизнес-правила в предназначенном для этого репозитарии.

Designer/2000, предназначенный для использования главным образом с Oracle 8, поддерживает все особенности данной СУБД, включая объектные типы данных (CLOB, Arrays, вложенные таблицы и др.), равно как и специфические особенности физической реализации базы данных Oracle. Для Oracle 7 и Oracle 8 это CASE-средство позволяет создать определения ролей, сгенерировать триггеры, реализующие бизнес-логику, которая описана в моделях, используемых при генерации базы данных, а также сгенерировать объекты для распределенных базы данных. Кроме того, с помощью Designer/2000 можно создавать физические модели и осуществлять обратное проектирование и для других СУБД - Oracle RDB, DB2, Microsoft SQL Server, Sybase, ODBC-источников данных, а также осуществлять обратное проектирование на основании DDL-сценариев, если они соответствуют стандарту ANSI SQL.

Весьма привлекательной особенностью Designer/2000 является возможность генерации форм Oracle Developer/2000, проектов Visual Basic, классов C++, отчетов Oracle Reports, приложений для Oracle Web Application Server.

ERwin (Computer Associates). ERwin разработан компанией Logic Works, которая в 1988 году была приобретена фирмой Platinum Technologies, а ее, в свою очередь, приобрела компания Computer Associates. Этот продукт в течение последних десяти лет занимает лидирующие позиции среди средств проектирования данных.

ERwin представляет собой специализированное средство проектирования данных. Его применение предполагает, что моделирование бизнес-процессов и потоков данных производится с помощью других продуктов (например, BPwin), с которыми можно осуществлять обмен сведениями о моделях.

ERwin не ориентирован на какую-то конкретную СУБД и поддерживает более 20 типов СУБД, включая СУБД всех ведущих производителей серверов баз данных (Oracle, Sybase, Microsoft, IBM, Informix), а также все популярные форматы настольных СУБД (включая dBase, Clipper, FoxPro, Access, Paradox), кроме, возможно, самых последних версий. Поэтому при использовании ERwin с последними версиями некоторых СУБД могут возникнуть проблемы (например, некоторые типы данных SQL Server 7.0 это CASE-средство поддерживает не совсем корректно). Тем не менее, ERwin остается одним из самых популярных в мире продуктов этого класса благодаря поддержке большого количества платформ, простоте интерфейса и, что немаловажно, поддержке специфических особенностей организации физической памяти наиболее популярных серверных СУБД. Например, для СУБД Oracle, Microsoft SQL Server, Sybase этот продукт позволяет изменять местоположение и параметры хранения индексов, почти для всех популярных серверных СУБД создавать кластеризованные индексы и для многих из них — указывать характеристики табличных пространств и сегментов отката.

Кластеризованный индекс — способ индексирования, при котором данные в таблице физически расположены отсортированными по значению поля, на котором построен индекс. Для каждой таблицы может быть создан только один кластеризованный индекс.

ERwin обладает встроенным макроязыком для написания в процессе логического проектирования не зависящих от СУБД шаблонов серверного кода, а также готовыми шаблонами для генерации триггеров, реализующих стандартные действия (например, каскадное удаление). При необходимости можно создавать и свои шаблоны триггеров, используя этот язык, причем каждая таблица может иметь свой набор шаблонов. При создании физической модели шаблоны преобразуются в код на процедурном расширении SQL того сервера, для которого создается физическая модель. Логическая и физическая модели ERwin хранятся в одном файле.

ERwin поддерживает обмен моделями с репозитарием Designer/2000 и Microsoft Repository, а также генерацию клиентских приложений для Visual Basic и PowerBuilder.

Помимо однопользовательской работы ERwin может выступать в роли клиентского приложения для другого CASE-средства — ModelMart, которое позволяет организовать коллективную разработку моделей данных, предоставляя для этой цели разделяемый репозитарий, хранящийся в одной из серверных СУБД.

Компанией Computer Associates выпущен новый продукт — ERwin Examiner, представляющий собой инструмент проверки баз данных и DDL-скриптов с целью выявления ошибок проектирования данных, сказывающихся на целостности данных и производительности сервера, таких, например, как ошибки нормализации, противоречивые ключи и т.д. В результате проверки ERwin Examiner предлагает способы устранения найденных ошибок, генерируя соответствующие DDL-скрипты.

PowerDesigner (Sybase). PowerDesigner (бывший S-Designor, принадлежавший компании PowerSoft) представляет собой инструмент, в состав которого входят средство

создания концептуальных (то есть логических) моделей, средство создания физических моделей и средство объектно-ориентированного моделирования, используемое при генерации клиентских приложений. Средство создания физических моделей представляет собой отдельный продукт — PowerDesigner PhysicalArchitect. В состав продукта PowerDesigner DataArchitect входят средства создания концептуальных и физических моделей, в состав PowerDesigner Developer — средства объектно-ориентированного моделирования и создания физических моделей, а в состав PowerDesigner ObjectArchitect — все три средства.

Физические и концептуальные модели в PowerDesigner DataArchitect хранятся в разных файлах, однако возможна генерация как физической модели на основе модели концептуальной, так и наоборот.

Помимо серверных СУБД производства Sybase (Adaptive Server Enterprise 12.0, Sybase SQL Anywhere) PowerDesigner DataArchitect способен работать с любыми ODBC-источниками. Как и ERwin, он поддерживает генерацию триггеров серверных СУБД, осуществляющих стандартную обработку событий, связанных с нарушениями ссылочной целостности.

PowerDesigner Developer и PowerDesigner ObjectArchitect могут генерировать код клиентских приложений для PowerBuilder, а также классы Java и компоненты JavaBeans. Возможно и обратное проектирование диаграмм классов из исходных текстов Java, байт-кодов и архивов Java. Поддерживается также генерация кода Web-приложений и объектов для Sybase Enterprise Application Server на основе физической модели.

PowerDesigner DataArchitect может импортировать логические и физические модели ERwin.

PowerDesigner DataArchitect может хранить свои модели данных в коллективно разделяемом репозитарии, управляемом с помощью средства PowerDesigner MetaWorks и доступном как дополнительный модуль в составе любого из перечисленных выше продуктов.

ER/Studio (Embarcadero Technologies). ER/Studio менее известен в нашей стране, чем ERwin и PowerDesigner DataArchitect. Однако возможности этого продукта также заслуживают внимания.

По своему назначению этот продукт сходен с ERwin — он представляет собой специализированное средство проектирования данных и не содержит в своем составе инструментов для объектно-ориентированного моделирования или моделирования бизнес-процессов. Список поддерживаемых СУБД у этого продукта достаточно широк и включает все наиболее популярные серверные и настольные СУБД. В отличие от ERwin последняя версия этого продукта корректно поддерживает новые типы данных SQL Server.

ER/Studio поддерживает написание макросов на SAX Basic (клон Visual Basic for Applications). Этот язык позволяет создавать макросы для выполнения однотипных операций, например добавления стандартных полей к вновь создаваемым сущностям. С помощью этого же языка можно генерировать стандартные триггеры и хранимые процедуры для вставки, удаления, изменения записей. Код на этом языке можно даже отлаживать и обращаться к свойствам сущностей для конструирования серверного кода. Однако, в отличие от ERwin, ER/Studio не позволяет добавить к каждой таблице свои шаблоны триггеров или просмотреть код конкретного триггера в процессе разработки модели — чтобы получить код одного триггера, нужно сгенерировать скрипт для всей модели.

Модели ER/Studio можно сохранить не только в виде DDL-скрипта, но и в формате XML. Можно также создать репозитарий для их хранения в любой серверной СУБД. ER/Studio может импортировать модели ERwin, но при импорте теряются связи шаблонов серверного кода с конкретными таблицами, и не все макросы ERwin корректно преобразуются в макросы SAX Basic.

ER/Studio позволяет сгенерировать Java-классы для клиентских приложений. ER/Studio является COM-сервером, что позволяет использовать его в других приложениях, предоставляя им возможность просмотра и редактирования моделей данных, а также создавать другие решения на его основе.

System Architect (Popkin Software). System Architect 2001 представляет собой универсальное CASE-средство, позволяющее осуществить не только проектирование данных, но и структурное моделирование. Средство проектирования данных и создания ER-диаграмм является одной из составных частей этого продукта.

Этот продукт поддерживает СУБД практически всех ведущих производителей, включая Oracle (Oracle 8), Sybase, DB2, SQL Server, IBM (AS400, DB2), Informix, Sybase, Access, dBASE, Paradox и др.

В процессе логического моделирования можно проверить модель на соответствие правилам проектирования данных (например, на соответствие ее первой, второй или третьей нормальным формам). При генерации DDL-скрипта можно сгенерировать триггеры (в том числе и нестандартные).

Все компоненты System Architect позволяют документировать процесс работы над проектом, включая техническое задание, план тестирования и др.

Модели System Architect 2001, как и в случае других CASE-средств, можно сохранять в репозитарии. Однако в отличие от традиционных репозитариев, обладающих более или менее стандартной структурой хранимых данных, репозитарий System Architect является настраиваемым — к сохраняемым объектам можно добавлять дополнительные свойства, определенные пользователем.

System Architect обладает встроенным Visual Basic for Application, что позволяет создавать разнообразные решения на базе этого продукта, включая автоматическую генерацию моделей и проектной документации.

System Architect 2001 позволяет генерировать код клиентских приложений для Visual Basic, Delphi и PowerBuilder (на сегодняшний день это практически единственное CASE-средство, поддерживающее генерацию кода Delphi), классы C++, а также код и текстовые экранные формы COBOL.

Visible Analyst (Visible Systems Corporation). Visible Analyst — весьма популярный продукт компании Visible Systems Corporation. Широко известны также ранее производимые этой компанией CASE-средства EasyER и EasyCASE — предшественники Visible Analyst.

Этот продукт выпускается в трех редакциях: Visible Analyst DB Engineer, который включает средства проектирования данных, Visible Analyst Standard, который кроме проектирования данных позволяет осуществлять структурное моделирование, и Visible Analyst Corporate, который помимо указанных выше возможностей позволяет осуществлять также объектно-ориентированное моделирование.

Visible Analyst поддерживает довольно широкий спектр СУБД с точки зрения генерации серверного кода, включая Oracle 7, Sybase SQL Server (System 10 и 4.x); Informix, DB2, Ingres. Для Informix и DB2 указанный продукт позволяет генерировать DDL-скрипты, учитывающие специфические особенности организации физической памяти наиболее популярных серверных СУБД, такие как управление табличным пространством, размером экстентов, режимами блокировки данных, степенью заполнения данными (fill factor), а также создавать кластеризованные индексы и генерировать триггеры для выполнения стандартных операций. Из этих же СУБД можно производить непосредственно обратное проектирование. Помимо этих двух СУБД обратное проектирование можно производить также из DDL-скриптов, сгенерированных для других СУБД, а также на основе кода COBOL.

Модели Visible Analyst можно сохранять в многопользовательском репозитарии, созданном в одной из серверных СУБД, Visible Analyst позволяет на основе созданных моделей генерировать код для Visual Basic, C++ и COBOL.

Отметим, что продукты семейства Visible Analyst имеют относительно невысокую стоимость, что также является их весьма привлекательной чертой.

Visio Enterprise (Microsoft). Продукт под названием Visio, приобретенный в январе 2000 года корпорацией Microsoft вместе с его разработчиком - компанией Visio Corporation, позиционировался на рынке как одно из самых популярных средств создания схем и диаграмм. Visio 2000 Enterprise — содержит в своем составе полноценное CASE-средство.

Visio Enterprise позволяет производить прямое и обратное проектирование данных, преобразовывать логическую модель в физическую. Этим средством поддерживаются все ODBC- и OLE DB-источники данных. С его помощью можно создавать триггеры для стандартной обработки нарушений ссылочной целостности в случае, если DDL-скрипт создается для Microsoft SQL Server, и серверные ограничения, если скрипт создается для другой СУБД. Visio при генерации скриптов позволяет указывать параметры организации физической памяти Oracle, Informix, Microsoft SQL Server, DB2 и некоторых других СУБД.

Visio включает средства объектно-ориентированного моделирования и генерации кода приложений Visual Basic 6, а также классов C++ и Java. Модели Visio можно сохранять в Microsoft Repository.

Visio, в отличие от специализированных средств проектирования данных, не обладает скриптовым языком, позволяющим создавать серверный код, не связанный с конкретной СУБД. При использовании этого продукта такой код нужно создавать на этапе физического проектирования в уже созданном скрипте. Стоимость Visio Enterprise по сравнению с ERwin или PowerDesigner DataArchitect невысока, тем более что Visio в целом представляет собой продукт более широкого назначения, нежели другие рассмотренные выше средства проектирования данных. К тому же этот продукт является сервером автоматизации, обладает весьма обширной объектной моделью и встроенным средством разработки — Visual Basic for Applications, что позволяет, в частности, создавать на его базе разнообразные решения, в том числе и автоматизировать разработку моделей данных.

6.3. Процесс создания базы данных

Структура базы данных является моделью предметной области. Она должна ее точно представлять и удовлетворять ее требованиям. Необходимо, чтобы процесс проектирования поддерживался всеми функциональными подразделениями предприятия, которые обязаны описать и определить элементы данных с точки зрения управляющего и пользователя.

В функции АБД входит устранение всех противоречий и двусмысленностей в определении данных. Качество проекта базы данных зависит от качества определения элементов данных и их взаимосвязей. Фактически процесс проектирования — это описание предприятия в терминах его наиболее важных объектов и внутренних связей.

Эффективному решению указанных задач способствует словарь данных (СД).

6.3.1. Словарь данных БД

Одно из главных назначений системы с базой данных — возможность создания условий для коллективного использования данных. Не менее важно предоставить пользователям достоверные данные. Наиболее удачным решением задачи обеспечения достоверности, минимальной избыточности и контроля использования данных является применение словаря данных. Такое решение, как правило, упрощает разработку и повышает эффективность системы.

На первом этапе проектирования базы данных необходимо собрать сведения о предметной области, в том числе о назначении, способах использования и о структуре данных, а, по мере развития проекта, осуществлять централизованное накопление информации о концептуальной, логической, внутренней и внешних моделях данных. Словарь данных является как раз тем средством, которое позволяет при проектировании, эксплуатации и развитии базы данных поддерживать и контролировать информацию о данных.

При сборе информации о данных следует:

- установить правила присвоения имен элементам,
- добиться однозначного толкования различными подразделениями назначения источников и соглашений по присвоению имен,
- сформулировать приемлемые для всех пользователей описания элементов данных и выявить синонимы.

Словарь данных содержит информацию об источниках, форматах и взаимосвязях между данными, их описания, сведения о характере использования и распределении ответственности. Он уже сам по себе является базой «данных о данных», руководством по базе данных.

Проектировщик базы данных рассматривает различные характеристики данных. На ранней стадии проектирования прежде всего готовятся описания элементов данных на естественном языке. Эти описания или определения должны быть точными, недвусмысленными и согласованными. На этой стадии разработки текстуальных описаний данных проектировщик абстрагируется от способа их физического представления в базе данных. В частности, ему не следует определять, как хранить данные — в упакованном, символьном или каком-либо другом виде.

Накопление информации о данных в словаре данных целесообразно начинать уже на самой ранней стадии проектирования. В процессе работы разработчики выясняют у пользователей, какой должна быть система, какие данные будут входными, какого рода информацию они хотят получить из системы, вводя имена элементов данных, например «номер счета», «остаток» или «процент» в банковской системе. При этом обе стороны должны трактовать используемые термины однозначно, иначе может случиться так, что разработанная система не будет удовлетворять требованиям пользователей.

Поэтому второе важное назначение словаря данных — обеспечить эффективное взаимодействие между различными категориями разработчиков и пользователей.

Рассмотрим следующий пример. В банковской системе одним из центральных элементов данных является «остаток». Каков остаток на данном счете? Для большинства неспециалистов ответ очевиден. Однако в главном файле счетов соответствующей системы в записи по одному счету может храниться до двадцати пяти полей, в именах которых присутствует термин «остаток». Поэтому важно, чтобы и пользователь и разработчик представляли себе, какой именно остаток имеется в виду: «остаток на счете на начало дня», «остаток на счете на конец вчерашнего дня», «фактический остаток» или «остаток на сберегательной книжке». «Остаток на сберегательной книжке» увеличивается сразу после того, как клиент делает вклад, но если вклад сделан с помощью чека, то «фактический остаток» увеличится только после оплаты чека. На самом деле существует гораздо больше различных полей «остаток». Словарь данных может использоваться для централизованного накопления информации обо всех элементах данных и для обеспечения эффективного взаимодействия между всеми участниками проекта.

Администрация большинства предприятий практически не осуществляет управления ресурсами данных из-за отсутствия единой точки зрения по этому вопросу.

Для управления ресурсами данных необходимо централизованное накопление информации о них. Только в этом случае администрация может управлять их использованием.

Таким образом, два важнейших назначения словаря данных состоят:

- в централизованном ведении и управлении данными как ресурсом на всех этапах проектирования, реализации и эксплуатации системы,
- в обеспечении эффективного взаимодействия между всеми участниками проекта.

6.3.2. Словарь данных и система управления базами данных

Словарь данных может быть полезен и в тех системах, где концепция базы данных не применяется. Он является средством централизованного накопления описаний данных: назначения, взаимосвязей с другими данными, источников, лиц, ответственных за ведение. В системе с базой данных словарь хранит информацию о базе данных, а в файловой — описания данных, содержащихся в файлах. Возможно одновременное использование словаря данных для прикладных программ, работающих под управлением СУБД и без СУБД.

Для создания базы данных словаря и управления ею должны применяться соответствующие программные средства, называемые словарем данных.

Пакет программ словаря данных может интегрироваться с системой управления базами данных или быть независимым.

Интегрированный словарь данных представляет собой часть пакета программ СУБД, а независимый словарь данных — отдельный пакет программ, служащий дополнением к СУБД.

В системе с интегрированным словарем данных описания данных существуют в единственном экземпляре и хранятся в этом словаре, поэтому необходимо всякий раз перед выполнением программы проверять достоверность используемых в ней описаний.

В системе с независимым словарем данных описания данных могут либо извлекаться из него, либо обеспечиваться другим образом, в этом случае подобная проверка не всегда обязательна.

Оба типа словарей данных, интегрированный и независимый, имеют свои преимущества и недостатки.

Преимущества интегрированной системы словаря данных:

• отсутствует дублирование описаний данных в СУБД и словаре данных, это сокращает вероятность появления ошибок в результате отказов при обновлении дублированной информации;

• словарь данных обладает возможностью доступа к базе данных, одним из его потенциальных применений может быть трассировка доступа к базе данных для сбора статистических данных с целью повышения производительности системы;

• будучи интегрированным с СУБД, словарь данных может служить более мощным средством контроля, поскольку как проектировщики, так и пользователи вынуждены его использовать в качестве инструмента документирования и настройки данных.

Преимущества независимого словаря данных:

• реализация сопряжена с меньшим риском попасть в рамки ограничений СУБД, чем у интегрированного словаря данных, кроме того, она проще, поскольку нет необходимости учитывать все особенности СУБД;

• описания всех данных могут вводиться постепенно, в то время как интегрированному словарю они должны быть предоставлены одновременно.

6.3.3. Идеальный словарь данных. Требования и организация

Приведем перечень требований к описанию данных, которым должен удовлетворять словарь данных. Отметим, что эти требования выполняются современными пакетами программ словаря данных не обязательно в полном объеме.

Концептуальная модель. В процессе разработки концептуальной модели необходима информация:

- об объектах, о представляющих их элементах данных или атрибутах и о взаимосвязях элементов,
- о текущем или планируемом использовании данных отдельными подразделениями и отдельными пользователями,
- о частотных характеристиках обращения к данным,
- нужны также текстуальные описания назначения и использования данных.

Элементы данных, объекты и взаимосвязи следует снабжать соответствующими метками с указанием:

- версии,
- статуса (планируемый, испытываемый, действующий и др.),
- текстуального описания,
- синонимов (других объектов словаря с отличной меткой, но имеющих тот же смысл),
- членства (составной частью каких объектов является данный элемент или какие объекты на него ссылаются),
- группирования элементов данных с выделением ключевых элементов.

Логическая модель. По логической модели базы данных в словаре данных должна храниться следующая информация:

- о группировании элементов данных с указанием ключевых элементов (возможно, выделяемые группы будут подмножествами групп, специфицированных в концептуальной модели),
- об используемой модели данных,
- о взаимосвязях групп данных в рамках применяемой модели,
- о внешних моделях, поддерживаемых логической моделью (различные логические пути доступа к данным),
- о логических транзакциях, программах и модулях,
- сведения о связях между логическими транзакциями, программами и модулями,
- а также таблицы соответствия между транзакциями, программами и модулями с одной стороны и подразделениями и функциями, которые они обслуживают, — с другой,
- для программ и транзакций необходимо специфицировать язык программирования и тип (пакетная, используемая в режиме телеобработки) обработки.

Внутренняя модель. В словаре данных должна храниться информация о физическом представлении данных:

- длина (в символах),
- тип представления (битовая или символьная строка, упакованное десятичное число, число с плавающей точкой),
- точность (для числовых данных),

- выравнивание (влево, вправо),
- шаблон (только для вывода данных),
- правила проверки значения (константа, диапазон значений),
- алгоритм вывода (для вычисляемых данных),
- местоположение (последовательная позиция, на которой размещается элемент данных внутри агрегата данных),
- защита (код защиты для чтения и обновления),
- носитель (перфокарты, диск, лента, экран терминала),
- устройства, на которых размещается база данных,
- информация, управляющая разграничением доступа.

Идеальный словарь данных должен быть неотъемлемой составной частью всей системы обработки данных. За ввод данных в словарь ответственность несет АБД. Поскольку словарь данных является центральным звеном системы, необходимо постоянно поддерживать его копию, которая может использоваться для восстановления словаря после возникновения отказа всей системы и в случае непреднамеренного разрушения его рабочей версии. За сохранность словаря данных, как жизненно важной части системы с базой данных, полностью отвечает администрация базы данных.

Идеальный словарь данных должен также обеспечивать ряд дополнительных возможностей.

Средства выборки данных и генерации отчетов. Всякое средство обеспечения взаимодействия между вовлеченными в разработку и эксплуатацию системы обработки данных лицами, а также ее документирования должны обладать широкими возможностями выборки данных и подготовки отчетов. При этом должна гарантироваться обработка как регламентных, так и произвольных запросов. Отчеты могут содержать:

• перечни элементов в алфавитном или любом другом порядке;

• распечатку таблиц соответствия между элементами данных, использующими их программами и подразделениями, отвечающими за обеспечение правильности значений данных;

• описания данных для программ, написанных на базовых языках программирования, а также логические представления внешних моделей для прикладных программ.

Кроме того, в словарь данных необходимо включать информацию, позволяющую прослеживать влияние изменений в структурах данных на прикладные программы и транзакции. Весьма желательно создать возможность предварительных (стоимостных) оценок последствий планируемых изменений.

Сбор данных для ввода в словарь. В идеальном случае словарь данных является тем инструментом, который совместно используется АБД, прикладными программистами, пользователями, администрацией и любыми программными средствами, взаимодействующими с базой данных. Данные, хранящиеся в словаре, должны вводиться людьми (с помощью ориентированных на них языковых средств) или средствами программного обеспечения из программных описаний данных, из описаний данных СУБД и из процедурных предложений программы.

Информация, управляющая разграничением доступа. В словаре данных может храниться информация по разграничению доступа, определяющая, какие именно пользователи и каким образом могут обращаться к той или иной части базы данных. Эта информация может служить следующим целям:

• перед компиляцией программ или выдачей отчетов она позволяет проверить, нет ли противоречий между спецификациями выборки данных, заданными осуществляющим доступ лицом, и заданными АБД спецификациями;

• для предотвращения неправильного использования данных из некоторой части базы данных, совместно обрабатываемой многими пользователями, на основе этой информации должны быть введены строгие ограничения безопасности данных.

Если словарь данных применяется для разграничения доступа к базе данных, то и к нему доступ надо также разграничить. Следует строго ограничить круг лиц, которым разрешено модифицировать словарь данных. В отношении хранимой в словаре информации должен быть реализован режим секретности.

Поддержка обслуживающих программ. Поскольку словарь данных является центральной частью системы с базой данных, он должен поддерживать работу различных обслуживающих базу данных программ: редактирования, внесения изменений, реструктуризации баз данных, генерации, отчетов и т. д.

Генерация исходного текста программ. С помощью хранящейся в словаре данных информации об элементах, их физическом расположении, взаимосвязях с другими элементами, а также о логической и внутренней модели, можно генерировать исходный код описаний данных и ряд стандартных подпрограмм, например модулей доступа или ввода-вывода.

Непротиворечивость. Хранимая в словаре данных информация должна быть полной, правильно форматированной и соответствующим образом взаимосвязанной. Для обеспечения этих требований необходим тщательный контроль непротиворечивости вводимых в словарь данных. Механизм таких проверок основывается на хранящихся в словаре данных спецификациях отображений между концептуальной, логической, внешними и внутренней моделями.

Идеальный словарь данных должен:

1) поддерживать концептуальную, логическую, внутреннюю и внешние модели;

2) быть интегрирован с СУБД;

3) поддерживать различные версии хранимых описаний (тестовые, рабочие);

4) обеспечивать эффективный обмен информацией с СУБД (в идеальном случае привязка внешних и внутренней моделей должна происходить во время выполнения программ, при этом словарь данных использует рабочие версии описаний и осуществляет динамическое построение описания базы данных и программ);

5) инициализировать процесс реорганизации рабочей версии базы данных после изменения описания последней.

С помощью словаря данных любые изменения в программах автоматически отражаются в их хранимых описаниях. Это обеспечивается в том случае, когда словарь данных интегрирован с СУБД.

6.4. Создание концептуальной модели предметной области

Создание базы данных, которая удовлетворяла бы текущим и перспективным информационным потребностям предприятия, связано с необходимостью проектирования концептуальной модели предметной области. Проектирование концептуальной модели основано на анализе решаемых на этом предприятии задач по обработке данных.

Концептуальная модель включает описания объектов и их взаимосвязей, представляющих интерес в рассматриваемой предметной области и выявляемых в результате анализа данных (используемых как в уже разработанных прикладных программах, так и в тех, которые только будут реализованы).

При разработке концептуальной модели может быть полезным аппарат реляционного подхода, не зависящий от особенностей реализации: концептуальная модель предметной

области может быть впоследствии отображена на реляционную, иерархическую или сетевую модель данных.

Проектирование концептуальной модели предметной области составляет одну из главных задач администратора базы данных. В этой модели должны быть представлены объекты и их взаимосвязи.

Концептуальная модель учитывает требования к обрабатываемым данным многих прикладных программ, а не каждой в отдельности. Представление отдельного прикладного программиста называется внешней моделью. Концептуальная модель не зависит от прикладных программ используемой СУБД, технических средств, на которых базируется система, и от внутренней модели данных, реализуемой в физической памяти.

При проектировании концептуальной модели все усилия разработчика должны быть направлены в основном на структуризацию данных и выявление взаимосвязей между ними без рассмотрения особенностей реализации и вопросов эффективности обработки.

Итак, первый этап, - анализ данных, позволяющий собрать информацию об элементах данных и их взаимосвязях.

Сбор информации о данных, используемых в существующих прикладных программах, является трудоемкой задачей и требует непременного участия руководства. Разработчик (АБД) должен разработать план проведения обследования предприятия. С помощью вопросника или иного подобного средства ему нужно составить списки данных, необходимые работникам всех уровней управления (исполнительного, функционального и эксплуатационного). Причем на различных уровнях данные могут обрабатываться или накапливаться. Затем АБД предстоит проанализировать все направления использования данных на предприятии.

Сбор данных следует начинать с изучения существующих форм документов, отчетов, имеющихся файлов и программ. Основной вопрос, требующий первоочередного решения, - какие именно данные должны быть представлены в новой базе данных. При этом необходимо учитывать, что подлежащие хранению данные редко однозначно соответствуют данным, отображаемым в формах и отчетах.

Каждый руководитель или сотрудник предприятия, который пользуется данными или участвует в их подготовке, должен заполнить соответствующую анкету. При этом чрезвычайно важно, чтобы ответственность за заполнение анкет несло руководство, поскольку анкеты отражают перспективные информационные потребности предприятия. В процессе сбора исходных данных следует предусмотреть дополнительные формы проведения опроса, чтобы дать возможность сотрудникам дополнить первоначальные списки требуемых данных, особенно тем, кто еще не был опрошен. Это существенный момент: если опрашивались не все сотрудники организации, необходимо пересмотреть модель предметной области и выяснить, почему в списке опрашиваемых лиц имеются пробелы.

Анкеты должны содержать: имя объекта данных, имя элемента данных, описание, атрибуты, источники, уровни конфиденциальности, показатели важности, а также взаимосвязи между элементами и между объектами.

Заполняющий анкету должен включать в нее как можно больше сведений, хотя может оказаться, что один человек будет не в состоянии заполнить все ее статьи. Тем не менее, не следует полагаться на то, что незаполненные статьи впоследствии будут заполнены другими лицами. Полное и точное представление о данных предметной области может быть получено только в том случае, если на предложенные в анкете вопросы каждый пользователь даст исчерпывающие ответы.

Ниже приводится краткое содержание анкет. Здесь не предлагается их конкретный формат. Важно, чтобы были получены ответы на все перечисленные вопросы. Выбор формата может определяться используемым словарем данных или существующими на

предприятии стандартами определения данных. Окончательный формат анкет устанавливает АБД.

Содержание анкет.

1. Имя и описание объекта данных.

Указываются основное имя и синонимы. Примеры: «Счета», «Журнал регистрации продукции», «Форма учета счетов». Дается вербальное описание смыслового содержания имени, даже если его смысл представляется очевидным. В общих чертах описывается функциональное назначение и использование объекта в функциональных и структурных подразделениях предприятия, а также за их пределами.

2. Элементы данных.

Для каждого элементарного данного, входящего в конкретный объект, указывается:

а) его имя и описание. Перечисляются имена, акронимы и дается их расшифровка. Приводится полное вербальное описание элемента;

б) источник. Перечисляются источники элемента в структуре предприятия, например заказчик, внутренние документы, отдел сбыта;

в) атрибуты. Указываются тип значения атрибута (числовой, алфавитный, текстовый), единицы измерения (доллары, футы), а при необходимости и допустимые диапазоны значений (например, от 100 до 500);

г) использование элемента данных. Примеры: «Содержит сведения об адресе», «Используется для определения количества», «Используется в шкале платежей»;

д) ограничения безопасности/чувствительности. Перечисляются связанные с данным элементом ограничения, включая допущенных к нему лиц и разрешенный им вид обработки, например доступ, чтение и/или выдача;

е) степень важности. Указывается степень важности данного элемента. Она должна определяться значением элемента данных для реализации или расширения функций предприятия. Следует избегать негативных формулировок типа «Без этого элемента данных невозможно выполнить то-то». Рекомендуется приводить аргументы, основываясь на использовании элемента данных (пункт г);

ж) взаимосвязи элемента данных. Описываются способы совместного использования данного элемента с другими, не обязательно принадлежащими рассматриваемому объекту. Примеры взаимосвязей: номер детали_ наименование, код операции_ трудозатраты, номер заказа_номер поставки.

3. Продолжительность хранения и условия перевода в архив.

Указывается период времени, в течение которого должны храниться значения элемента данных, и способ хранения. По возможности также указывается основание для хранения (правительственные распоряжения, указания администрации предприятия).

Одновременно с проведением анкетирования АБД должен исследовать информационные потоки в аппарате управления, в канцелярии при обработке данных. Целью такого исследования является не столько проверка правильности заполнения анкет, сколько разработка модели предприятия. В результате АБД получает представление о документообороте предприятия, определяет пути и способы передачи данных.

Следующий и, вероятно, наиболее важный этап - анализ организации хранения данных, базирующийся на результатах анкетирования и исследования документооборота.

Этот анализ достаточно просто описывается, но не так легко выполняется. АБД разрабатывает графическую схему объектов и элементов данных, на которой указываются исходные данные, формирующие их подразделения или виды деятельности, результирующие данные и использующие их подразделения. Выявленный при анкетировании документооборот отражается на специальных схемах, целесообразно использовать возможности диаграмм потоков данных (нотация DFD, BPwin (CA)). Простейшая схема

данных показывает их движение от источника к конечному пользователю. В процессе разработки схемы данных АБД неизбежно встретится с противоречиями, ошибками и неточностями в исходных описаниях, которые он обязан обнаружить и устранить. Удобным средством при анализе данных может оказаться словарь данных.

АБД уточняет степень важности отдельных данных для конечного пользователя, сопоставляя выдвигаемые пользователем требования к объектам и элементам с реально существующими. При этом он должен обладать достаточными полномочиями для того, чтобы действительно управлять информационными ресурсами и вносить необходимые изменения в принятую схему обработки данных.

По завершении анализа АБД обязан обсудить с представителями различных подразделений выявленные связи между данными, их источники и требования пользователей. Необходимо, чтобы пользователи и АБД пришли к единому мнению относительно перспектив обработки данных на предприятии.

После этого АБД может приступить к разработке модели данных (концептуальной модели базы данных).

Сбор информации о данных для перспективных приложений. Сбор информации для планирования перспективного использования базы данных - одна из наиболее важных и сложных задач АБД. Обычно после ввода базы данных в эксплуатацию пользователи, оценив на практике ее значение для принятия решений и обработки информации, предъявляют более высокие требования к составу реализуемых функций, вносят предложения по введению новых перекрестных ссылок и улучшению операционных характеристик системы. Если основу проектирования составляют только текущие требования к базе данных, то это может затруднить реализацию новых. Для того чтобы подобные проблемы в будущем не возникали, АБД должен заранее учитывать возможные пути использования информации. Это достаточно трудно, но, тем не менее, АБД приходится выявлять неучтенные объекты и взаимосвязи, которые могут и не быть задействованы ни в каких функциях, и детально обсуждать их с пользователями. Информация о вероятных путях перспективного использования данных не только необходима для проектирования концептуальной и логической модели, но может оказать определенное влияние на выбор способа физической реализации базы данных. Выбор физической модели в особенности зависит от оценок будущих объемов информации.

6.5. Реляционные основы проектирования

Информационная модель предметной области более устойчива, чем способы выборки хранимой в базе данных информации. Один из эффективных методов разработки, так называемой, концептуальной модели предметной области - введение ряда понятий и концепций реляционной модели данных. Эти понятия и концепции применяются в процессе анализа информации о данных и их структуре, полученной у конечных пользователей. Это вовсе не означает, что концептуальная модель должна быть реализована с помощью реляционной системы управления базами данных. Концептуальная модель служит основой для создания логической модели, которая может быть реализована средствами любой СУБД.

Основное понятие, заимствованное из реляционной модели данных, вводимое при разработке концептуальной модели,— это нормализация отношений. В процессе нормализации элементы данных группируются в таблицы, представляющие объекты и их взаимосвязи.

Теория нормализации основана на том, что определенный набор отношений обладает лучшими свойствами при включении, модификации и удалении данных, чем все остальные наборы отношений, с помощью которых могут быть представлены те же данные.

Введение нормализации отношений при разработке концептуальной модели обеспечивает ее работоспособность. Это вовсе не означает, что ненормализованная концептуальная модель обязательно окажется неработоспособной. Дело в том, что ненормализованная модель может вызвать определенные трудности реализации прикладных программ, модифицирующих базу данных. Обнаружив отклонения от нормализованной схемы, АБД должен решить, насколько эти отклонения ухудшают характеристики базы данных при модификации.

Ненормализованная модель данных включает записи в том виде, в котором они используются прикладными программами. Первый шаг при нормализации заключается в образовании двумерной таблицы, содержащей элементы данных. Для этого практически нужно лишь исключить повторяющиеся группы. Например, если в декларацию заносится имя сотрудника, его идентификационный номер, имя его супруги и имена его детей (предполагается не более десяти), то сведения о нем могут быть представлены с помощью таблицы, состоящей из десяти строк и четырех столбцов. В каждой строке записывается имя сотрудника, его номер, имя супруги и имя одного из детей. Таким образом, в десяти строках таблицы будут находиться имена всех десяти детей. Исключение повторяющихся групп является предварительным этапом нормализации, после чего можно перейти к получению второй нормальной формы.

Второй шаг нормализации состоит в том, чтобы выделить ключи и зависящие от них атрибуты. Каждый кортеж отношения, находящегося в первой нормальной форме, полностью зависит от совокупности ключевых атрибутов. Для того чтобы привести отношение ко второй нормальной форме, нужно выделить группы атрибутов, зависящие от частей составного ключа. Эти группы могут образовать отдельные отношения (таблицы). Выделение из отношения, находящегося в первой нормальной форме, таких отношений, в которых неключевые атрибуты зависят только от ключа в целом, называется приведением ко второй нормальной форме

На третьем шаге нормализации следует выделить из отношений, находящихся во второй нормальной форме, те атрибуты, которые, хотя и зависят от ключа какого-либо отношения, тем не менее, могут существовать в базе данных независимо от остальных атрибутов этого отношения. Выделение атрибутов позволяет вводить их значения вне зависимости от взаимосвязей, в которых они участвуют.

В любой модели данных для представления объектов и их взаимосвязей необходимо некоторым образом сгруппировать элементы данных. При обработке групп элементов возникают три общих проблемы. Устранение этих проблем требует приведения отношений к одной из трех нормальных форм. Таким образом, процесс нормализации, выполняемой по определенным правилам, состоит в группировке элементов данных в ряде отношений.

Приведение отношений к первой, второй и третьей нормальной форме последовательно устраняет аномалии при включении, удалении и модификации записей соответствующей базы данных. Процесс нормализации позволяет проектировщику глубже понять семантику атрибутов и их взаимосвязей и упорядочивает проведение анализа данных.

Графическое представление. Известно, что подчас изображения служат более выразительным средством общения, чем речь. Для графического отображения процесса проектирования концептуальной модели целесообразно воспользоваться методологией создания информационной модели IDEF1X в среде ERwin (CA).

Концептуальная логическая модель данных описывает факты и объекты, подлежащие регистрации в будущей базе данных. Основными компонентами такой модели являются сущности, их атрибуты и связи между ними. Как правило, физическим аналогом сущности в будущей базе данных является таблица, а физическим аналогом атрибута - поле этой таблицы.

С логической точки зрения сущность представляет собой совокупность однотипных объектов или фактов, называемых экземплярами этой сущности. Физическим аналогом

экземпляра обычно является запись в таблице базы данных. Как и записи в таблице реляционной СУБД, экземпляры сущности должны быть уникальными, то есть полный набор значений их атрибутов не должен дублироваться. И так же, как и поля в таблице, атрибуты могут быть ключевыми и неключевыми.

На этапе концептуального логического проектирования для каждого атрибута обычно определяется примерный тип данных (строковый, числовой, BLOB и др.). Конкретизация происходит на этапе физического проектирования, так как различные СУБД поддерживают разные типы данных и ограничения на их длину или точность.

Связь с логической точки зрения представляет собой соотношение между сущностями, которое нередко может быть выражено обычными фразами, например: «Клиент размещает заказ» («A customer places an order» — этой фразой вполне можно описать связь, изображенную на рис.6.2), где существительными являются названия связанных между собой сущностей.

Подавляющее большинство средств проектирования данных позволяют создавать ER-диаграммы визуально, изображая сущности и соединяя их связями с помощью мыши. Интерфейс таких средств нередко настолько прост, что позволяет освоить логическое проектирование данных не только разработчику, но и пользователю-непрограммисту, если таковой участвует в проектировании данных как эксперт в предметной области.

Отметим, что на этапе концептуального логического проектирования можно описать поведение СУБД при нарушении правил ссылочной целостности, определяемых данной связью (например, при удалении сведений о заказчике, разместившем хотя бы один заказ, для связи, изображенной на рис. 6.1).

Для этого многие (но не все!) средства проектирования данных обладают языком шаблонов, не зависящим от СУБД, на котором можно описать алгоритм обработки подобной ситуации, например с помощью соответствующих триггеров или иных объектов базы данных, а также набором стандартных шаблонов, реализующих некоторые типовые алгоритмы подобной обработки (например, отказ от удаления с последующей генерацией диагностического сообщения, каскадное удаление дочерних записей и др.). В процессе создания физической модели эти шаблоны преобразуются в код на процедурном расширении языка SQL, применяемом в конкретной СУБД.

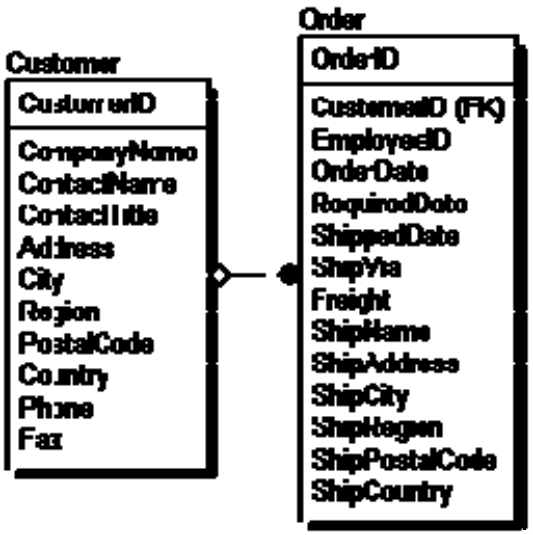

Рис. 6.1. Пример связи между сущностями концептуальной модели

Отметим, что концептуальная модель данных, как правило, не связана с конкретной СУБД и не учитывает технические особенности конкретных платформ, применяемых при ее последующей физической реализации.

Необходимо систематически исследовать назначение, а также входные и выходные данные для каждой прикладной программы, которая обращается к базе данных. При проектировании, прежде всего, должны учитываться требования конечных пользователей. Элементы данных, их взаимосвязи и элементы, служащие ключами, определяются на основе концептуальных требований. С их помощью АБД сможет разработать концептуальную модель предметной области.

6.6. Создание логической модели

Создавать логическую модель намного проще, если иметь точную и аккуратно построенную диаграмму ERD. Мы применяем прием проектирования сверху вниз или движения от диаграммы ERD в прямом направлении. Альтернативный подход, проектирование снизу вверх, предполагает построение модели путем применения правил нормализации данных.

Правила прямого преобразования модели ERD в логическую модель просты и понятны. Сущности, использованные в диаграмме ERD, становятся таблицами логической модели. Атрибуты превращаются в столбцы. Каждый первичный идентифицирующий атрибут становится первичным ключом (PK).

Две сущности, связанные отношением один - к одному (1:1), будут иметь один и тот же первичный ключ. Если же между ними существует не идентифицирующее отношение один – ко многим (1:M), то экземпляр одной сущности может существовать и без связанного с ним экземпляра другой сущности. В базе данных может находиться запись для служащего, в которой будет отсутствовать название его должности.

Несколько сложнее происходит прямое преобразование идентифицирующего отношения 1:M. В базе данных таблицы A и B связаны между собой идентифицирующим отношением 1:M. Строка таблицы B не может существовать без соответствующей строки таблицы A. Первичный ключ идентифицирующей сущности A становится внешним ключом слабой сущности B. Слабая сущность располагается в таком отношении на стороне многие и не может существовать без связанной с ней сущности, расположенной на стороне один. Для формирования кандидата на роль составного первичного ключа можно использовать комбинацию внешнего ключа и идентификатора слабой сущности.

Бинарные отношения многие - ко многим (то есть отношения между двумя сущностями, связанными между собой как M:N) порождают три таблицы в логической модели. Каждая сущность превращается в отдельную таблицу, и еще одну таблицу образует само отношение. Это таблица перекрестных ссылок или ассоциаций. Таблица ассоциаций наследует первичные ключи обеих таблиц, участвующих во взаимодействии M:N первоначально в качестве внешних ключей. В дальнейшем они могут стать ее составным первичным ключом.

Тернарные отношения (в которых три сущности связаны между собой отношением многие - ко многим - ко многим, M:N:O), порождают четыре таблицы аналогично тому, как это происходит в случае бинарного отношения многие – ко многим. При этом возможным кандидатом на роль первичного ключа таблицы ассоциаций может стать композиция первичных ключей всех трех таблиц, участвующих в отношении M:N:O.

Рассмотрим отображение концептуальной модели на логическую модель данных поддерживаемую конкретной СУБД.

При разработке логической модели базы данных прежде всего необходимо решить, какая модель данных наиболее подходит для отображения конкретной концептуальной модели предметной области. Коммерческие системы управления базами данных поддерживают одну из известных моделей данных или некоторую их комбинацию. Известны три наиболее распространенных модели данных реляционная, иерархическая и сетевая.

Отображение на реляционную модель данных. Реляционная модель данных состоит из ряда отношений (таблиц). При отображении концептуальной модели на логическую модель необходимо определить отношения и их атрибуты. Отдельные атрибуты этих отношений являются их первичными ключами. Каждый прямоугольник представляет отношение и содержит присущие ему атрибуты.

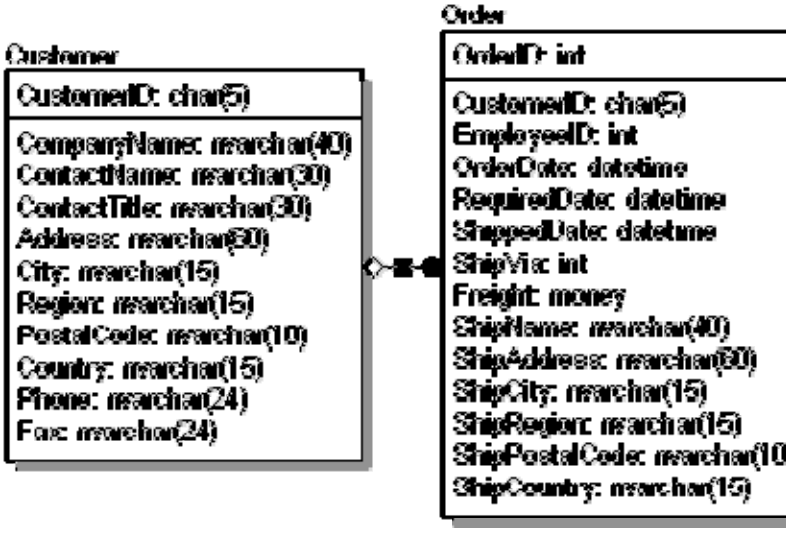

Рис. 6.2. Пример отображения простейшей концептуальной модели данных на логическую модель РСУБД

Отображение концептуальной модели данных на реляционную проводится относительно просто. Каждый прямоугольник концептуальной модели отображается в одно отношение, которое отражает представление пользователя в удобном для него табличном формате. Простота отображения связана с тем, что при разработке концептуальной модели использовался реляционный подход. Пример логической модели данных, поддерживаемой СУБД SQL Server2000, соответствующую приведенной выше концептуальной логической модели, представлен на рис.6.2.

Отображение на иерархическую модель данных. Преобразование концептуальной модели в логическую иерархическую модель данных сложнее, поскольку при этом существует кажущаяся свобода выбора конкретных решений. Как правило, в таких случаях единственно верного решения быть не может. Однако преобразование модели можно разбить на этапы и определить для каждого из них критерии выбора решения. Различают основные этапы.

А. Вывод обобщенной иерархической модели, в которой не учитываются ограничения, накладываемые используемой СУБД (устранение транзитивности, выявление взаимосвязей типа «исходный — порожденный», устранение множественного родительства).

Б. Трансформация полученной модели с учетом ограничений, накладываемых конкретной СУБД.

В. Модификация трансформированной модели с учетом «очевидных» соображений, влияющих на производительность.

- Исходный сегмент, обладающий только одним порожденным типом сегмента, может с ним объединяться.

- Следует ли сохранить логическую взаимосвязь или объединить логически исходный и логически порожденный сегменты?

Г. Упрощение имен ключей (имена ключей порожденных сегментов можно упростить, удалив те части имени, которые встречаются в имени исходного сегмента).

Д. Реализация взаимосвязей, не отображенных в логической модели, но на самом деле существующих.

Отображение на сетевую модель данных. Отображение концептуальной модели на сетевую модель данных представляет собой сложный процесс, имеется ряд альтернатив, нет очевидного наилучшего решения и требуется выполнение следующих основных этапов.

А. Формирование обобщенной сетевой модели данных, в которой не учитываются ограничения, накладываемые используемой СУБД (определение взаимосвязей «владелец - член», устранение множественного владения).

Б. Трансформация обобщенной сетевой модели с учетом ограничений, накладываемых конкретной СУБД.

В. Модификация трансформированной модели с учетом «очевидных» соображений, влияющих на производительность (в сетевой модели данных в отличие от большинства других моделей АБД должен указывать способы размещения записей, выбор способа размещения существенно влияет на эксплуатационные характеристики базы данных).

Г. Упрощение имен ключей.

Д. Реализация взаимосвязей, не отраженных в логической модели, но на самом деле существующих.

6.7. Создание физической модели

Физическое проектирование данных осуществляется на основе логической модели. Результатом этого процесса является физическая модель, содержащая полную информацию, необходимую для генерации всех необходимых объектов в базе данных. Для СУБД, поддерживающих системный каталог, физическая модель соответствует его содержимому.

Обычно на основании физической модели создается DDL -скрипт для создания объектов базы данных либо эти объекты создаются непосредственно из CASE-средства — подавляющее большинство современных средств такого класса поддерживает различные механизмы доступа к данным и может выступать в роли клиентского приложения, инициирующего выполнение DDL-скриптов (DDL (Data Definition Language) — язык определения данных, подмножество языка SQL). Пример скрипта для создания объектов базы данных, сгенерированный на основании логической модели (рис.6.2) с помощью одного из рассмотренных ниже CASE-средств, приведен в листинге 6.1.

Листинг 6.1

```
CREATE TABLE Customer (
    CustomerID          char(5) NOT NULL,
    CompanyName         nvarchar(40) NOT NULL,
    ContactName         nvarchar(30) NULL,
    ContactTitle        nvarchar(30) NULL,
    Address             nvarchar(60) NULL,
    City                nvarchar(15) NULL,
    Region              nvarchar(15) NULL,
```

```
    PostalCode              nvarchar(10) NULL,
    Country                 nvarchar(15) NULL,
    Phone                   nvarchar(24) NULL,
    Fax                     nvarchar(24) NULL
)
ALTER TABLE Customer
    ADD PRIMARY KEY (CustomerID)

CREATE TABLE Order (
    OrderID                 int IDENTITY,
    CustomerID              char(5) NULL,
    EmployeeID              int NULL,
    OrderDate               datetime NULL,
    RequiredDate            datetime NULL,
    ShippedDate             datetime NULL,
    ShipVia                 int NULL,
    Freight                 money NULL,
    ShipName                nvarchar(40) NULL,
    ShipAddress             nvarchar(60) NULL,
    ShipCity          nvarchar(15) NULL,
    ShipRegion              nvarchar(15) NULL,
    ShipPostalCode          nvarchar(10) NULL,
    ShipCountry             nvarchar(15) NULL,
    FOREIGN KEY (CustomerID)
    REFERENCES Customer
)
CREATE INDEX XIF1Order ON Order ( CustomerID)
ALTER TABLE Order   ADD PRIMARY KEY (OrderID)
create trigger tU_Customer on Customer for UPDATE as
begin
  declare @numrows int, @nullcnt int,
 @validcnt int, @insCustomerID char(5),
 @errno int, @errmsg varchar(255)

  select @numrows = @@rowcount
  if update(CustomerID)
  begin
if exists ( select * from deleted,Order where
Order.CustomerID = deleted.CustomerID )
begin
select @errno = 30005,
@errmsg = 'Cannot UPDATE Customer because Order exists.'
goto error
```

```
end
  end
  return
error:
raiserror @errno @errmsg
rollback transaction
end
```

Физическое проектирование может включать и дополнительную «ручную» работу по доведению базы данных или скрипта для ее генерации до «товарного» вида. Например, нередко в скрипт также включаются SQL-предложения для заполнения некоторых таблиц, данные которых более или менее постоянны, таких, например, как список субъектов Российской Федерации или справочник телефонных кодов различных стран, а также нестандартные триггеры и процедуры.

В последнее время в техническое задание на разработку приложений нередко включается полное описание физической модели данных или ее части, с которой должно иметь дело разрабатываемое приложение. Подавляющее большинство современных средств проектирования данных предоставляют возможность не только документировать модель, но и создавать по ней отчеты, содержащие те или иные сведения о модели, в том числе сведения, необходимые для разработки приложений.

6.8. Реализация базы данных

Физическая реализация отличается от физического проекта или модели базы данных. Физическая модель описывает базу данных в конкретном рабочем окружении, которое включает окончательно выбранную систему управления базой данных, определенную конфигурацию технических средств и сетевого окружения, а также заданный уровень нагрузки, обусловленной поиском и обновлением данных. Физическая реализация претворяет физический проект в жизнь. Развернутая база данных содержит объекты (таблицы, представления, индексы), которые соответствуют объектам физической модели.

Процесс физической реализации базы данных представляется в виде последовательности шагов, которую можно использовать в качестве плана действий.

1. Выбор сервера.

Модели, предназначенные для анализа использования данных (Data Use Analysis) и анализа объемов данных (Data Volume Analysis), позволяют выбрать сервер с нужными параметрами быстродействия процессора и объема дисковой памяти. Такой сервер должен обеспечить нормальное функционирование базы данных в течение нескольких лет. Модель использования данных позволяет визуально выделить важнейшие процессы, использующие информацию БД. Затем подсчитывается среднее и максимальное количество операций чтения и записи. Этот анализ позволяет сформулировать требования к производительности процессора и объему оперативной памяти. Модель, построенная для анализа объемов данных, представляет собой модификацию диаграммы сущность-связь (entity relationship diagram - ERD). Она помогает рассчитать необходимый базе данных объем памяти на жестком диске.

2. Создание базы данных.

Для оценки объемов пользовательских данных и размера журнала транзакций на этом этапе опять можно воспользоваться моделью использования данных. Такая модель дает грубую оценку потребностей в памяти; ее следует преобразовать в рекомендации по назначению исходных размеров файлов баз данных.

3. Создание объектов базы данных.

Существует два варианта генерации таблиц, представлений, индексов, ограничений, хранимых процедур и триггеров, составляющих оперативную базу данных. Первый вариант предусматривает применение физической модели для написания сценариев SQL, которые можно будет в дальнейшем исполнять. Альтернативный способ реализации первого варианта заключается в использовании графического и программного интерфейса Enterprise Manager для непосредственного создания объектов базы данных. Второй вариант основан на применении специального программного инструментария CASE, такого как ERwin, Visio 2000, который существенно облегчает моделирование. Средства CASE могут генерировать все необходимые сценарии автоматически, используя информацию соответствующих моделей. Мастер генерации позволяет либо сгенерировать текстовый файл сценария на языке DDL (Data Definition Language), либо создать новую базу данных, либо построить модель, описывающую уже существующую базу данных.

4. Загрузка данных.

Как подойти к загрузке информации в спроектированную базу данных? Ответ зависит от того, из какого источника берутся сведения и в каком объеме. Если первоначально при создании базы данных не требуется загружать в нее информацию, то остается сосредоточиться лишь на тех схемах сбора данных в базу, которые предполагается использовать. К их числу относятся формы для ввода информации и автоматические программы сбора сведений, наподобие тех, которые применяются в промышленных автоматизированных системах. Вероятнее всего, придется импортировать данные из плоских файлов с разделителями в виде запятой или же производить преобразования сведений, поступающих в базу, из других систем. Если необходимо загружать информацию из плоских файлов с разделителями в виде запятой, то лучшим вариантом реализации может стать применение программы копирования массивов (bcp). Программа bcp обеспечивает минимальные накладные расходы и максимальную скорость загрузки данных, поскольку она не генерирует записи в журнал транзакций. Поэтому не надо беспокоиться об откате транзакций, обновлении индексов или о проверке выполнения условий ограничений. Однако в тех случаях, когда необходимо импортировать данные, требующие выполнения определенных преобразований (например, реорганизации или реструктуризации), а также когда загружаемые данные приходят из другой базы данных или системы, в которой нет баз данных, тогда не обойтись без службы Data Transformation Services, которая входит в состав SQL Server. Если не требуется производить какую-либо трансформацию данных, то DTS обеспечит такую же быструю загрузку данных, как и bcp. К тому же DTS позволяет сохранить сценарии и использовать их в дальнейшем в качестве основы для выполнения более сложных преобразований и загрузки данных. Для загрузки данных и работы с ними можно использовать и продукты независимых компаний. Выбирая средство для загрузки данных, нужно учитывать следующее:

- оно должно быть знакомым,
- должно оперативно обеспечивать загрузку,
- оптимально использовать ресурсы системы (центральный процессор, оперативную память, сетевую карту, жесткий диск и т.п.).

5. Создание и запуск сценариев безопасности.

Написание сценариев безопасности относится к тому типу задач, которые приходится выполнять вручную. В качестве руководства по построению сценариев безопасности SQL Server можно использовать средства создания системы безопасности Enterprise Manager, которые могут сгенерировать все необходимые сценарии.

6. Формирование задач резервного копирования и прочих задач сопровождения.

Теперь, когда база данных создана и уже заполнена информацией, необходимо реализовать план предупреждения возникновения аварийных ситуаций. Для настройки

расписаний снятия копий и восстановления всех пользовательских и системных баз данных можно воспользоваться входящим в состав SQL Server мастером планирования сопровождения баз данных (Database Maintenance Plan Wizard). Этот мастер может помочь сформировать задачи, связанные с реорганизацией страниц данных и индексов, обновлением статистики, восстановлением пространства, которое не используется файлами баз данных, с проверкой целостности баз данных, а также составлением отчетов об использовании разнообразных утилит.

7. Выполнение различных задач.

Если предстоит проведение репликаций, придется продумать, какой вид следует избрать: репликации при помощи мгновенных снимков, репликации транзакциями или же репликации слиянием. (Более подробно об этом рассказано в статье Тэда Дейли и Боба Пфайфа "Зона слияния", опубликованной в журнале Windows 2000 Magazine/RE, декабрь, 1999 год.)

- Репликации при помощи мгновенных снимков состояния обеспечивают отсылку в указанную базу данных копий, которые снимаются регулярно, через заданный интервал. Это оптимальный вариант при относительно статичных наборах данных, которые следует обновлять от случая к случаю. Этот вид репликации очень удобно использовать для работы с выездными агентами, которые лишь изредка подключаются со своих переносных компьютеров к центральной БД, чтобы загрузить копии файлов данных.
- При репликации транзакциями копирование данных происходит по мере их изменения в исходной базе данных. Выбирать этот вид следует в том случае, когда изменения должны попадать в базу данных практически мгновенно, то есть когда есть возможность установить постоянное соединение между базами данных, которые являются, соответственно, источником и адресатом.
- Репликация слиянием позволяет вносить изменения в базы данных на месте, а затем отсылать их в исходную базу данных, где и осуществляется слияние. Этот вид лучше всего использовать, если в организации работает много выездных агентов, которые вносят изменения в данные, он очень эффективен для горизонтально секционированных файлов, когда секционирование проводится по географическому признаку.

Какой вид репликации лучше всего отвечает потребностям решаемых задач? Выбор за разработчиком.

6.9. Методы хранения и доступа к данным

Одним из основных факторов влияющих на производительность программ, которые взаимодействуют с базой данных, является способ хранения и доступа к данным Обычно в дополнение к специализированным методам доступа в рамках внешней модели СУБД использует несколько методов доступа внутренней модели. Внутренняя модель - это модель физическая, внешняя же отражает представление пользователя (о базе данных). Некоторые методы доступа внутренней модели совпадают с методами доступа операционной системы. Большинство коммерческих СУБД позволяют проектировщику базы данных варьировать параметры физической организации. Для того чтобы база данных обеспечивала эффективное хранение и доступ к данным, проектировщик должен знать методы доступа как внутренней модели (физической) и внешней (представления пользователя).

Интерфейсы между пользователем и базой данных. На практике увеличение числа операции физического ввода- вывода выполняемых при выборке данных для удовлетворения запроса пользователя, приводит к снижению производительности.

В процессе проходит несколько интерфейсов.

Интерфейс 1. Рассматривает описание представления пользователя и прикладной программы, а также условия безопасности и секретности. На основании представления пользователя определяется, к какой физической базе (базам) данных может осуществляться доступ. Кроме того, из описания физической базы (баз) данных известен используемый метод (методы) доступа внутренней модели. Разные реализации предоставляют различные возможности, однако в большинстве из них поддерживается несколько представлений пользователя о базе данных.

Интерфейс 2. В свою очередь СУБД использует методы доступа внутренней модели, которые в разных системах реализованы по разному: либо как специализированные методы доступа, предоставляемые СУБД, либо как сформированные из обобщенных методов доступа операционной системы. Примерами могут служить ISAM (индексно последовательный метод доступа) и BOAM (базисный прямой метод доступа).

Выполнение запроса пользователя обеспечивают СУБД, методы доступа внешней и внутренней моделей, а также методы доступа операционной системы.

Интерфейс 3. Методы доступа внутренней модели совместно с методами доступа операционной системы осуществляют доступ к записи (записям) физической базы данных и создают основу для нахождения нужной записи (записей) методом (методами) доступа, описывающими логические взаимосвязи между различными частями записи базы данных.

Методами доступа операционной системы, а также внешней и внутренней моделей, осуществляют выборку данных из физической базы (баз) данных и их передачу СУБД, которая определяет, какая часть данных может быть выдана пользователю, в каком формате и т.д.. АБД должен описать все эти характеристики для СУБД до начала реализации базы данных.

Производительность базы данных в значительной степени зависит от методов доступа внутренней и внешней моделей. Разработчику следует изучить возможности использования различных методов доступа конкретной закупаемой или арендуемой СУБД.

6.9.1 .Методы доступа внутренней модели (физической)

Различают следующие методы доступа внутренней модели:

- физический последовательный,
- индексно-последовательный,
- индексно-произвольный,
- инвертированный
- и посредством хеширования.

Для каждого из них зададим два критерия.

1. Эффективность доступа - величина, обратная среднему числу физических обращений, необходимых для осуществления логического доступа, т.е. запроса конкретной записи базы данных. Физические обращения обеспечивают удовлетворение запроса. Например, если для поиска нужной записи система обращается к двум записям, то эффективность доступа равна 0,5.

2. Эффективность хранения - величина, обратная среднему числу байтов поля вторичной памяти, требуемого для хранения одного байта исходных данных. Кроме исходных данных память занимают таблицы, управляющая информация, свободная область, резервируемая для расширений, и область, не используемая из-за фрагментации.

Физический последовательный. Значения ключей физических записей находятся в логической последовательности.

- В основном применяется для "дампа" и "восстановления"
- Может применяться как для хранения, так и для выборки данных
- Эффективность использования памяти близка к 100%

Физический последовательный метод доступа предполагает хранение физических записей в логической последовательности. При использовании в качестве носителя хранимых данных магнитной ленты, программист должен предоставлять системе физические записи в логической последовательности. В том случае, когда носителем служит память прямого доступа, система связывает физические записи таким образом, что они образуют логическую последовательность, даже если они не передавались ей в этой последовательности.

Эффективность доступа. В худшем случае для выборки нужной записи необходимо просмотреть все записи базы данных, а в лучшем достаточно выбрать следующую запись. Число записей, к которым необходимо осуществить доступ для выборки нужной, составляет в среднем половину общего числа записей, содержащихся в базе данных.

Эффективность хранения. Если носителем данных служит магнитная лента, программист должен предоставлять записи базы данных в логической последовательности. В результате память на магнитной ленте будет заполняться плотно. При использовании устройства прямого доступа каждой следующей представляемой физической записи система отводит следующий доступный участок физической памяти.

Индексно-последовательный. Значения ключей физических записей находится в логической последовательности, может применяться как для хранения, так и для выборки данных, в индекс значений ключей заносятся статьи наибольших значений ключей в блоках, наличие дубликатов значений ключей недопустимо.

Эффективность доступа зависит от числа уровней индексации, распределения памяти для размещения индекса, числа записей базы данных и уровня переполнения.

Эффективность хранения зависит от размера и изменяемости базы данных.

Оценим эти два метода доступа внутренней модели: физический последовательный и индексно-последовательный.

Эффективность доступа физического последовательного метода оставляет желать лучшего. Для выборки нужной записи требуется просмотреть все предшествующие ей записи базы данных.

Метод доступа, при использовании которого до осуществления доступа к собственно записям базы данных проверяются значения ключей, называется индексно-последовательным.

Индексно-произвольный.

- Значения ключей физических записей необязательно находятся в логической последовательности.
- Хранение и доступ к индексу могут осуществляться с помощью индексно-последовательного метода доступа.
- Индекс содержит статью для каждой записи базы данных. Эти статьи упорядочены по возрастанию. Ключи индекса сохраняют логическую последовательность. Если же они эту последовательность не сохраняют, доступ к индексу осуществляется посредством алгоритма хеширования. Записи базы данных могут быть и не упорядочены по возрастанию ключа.

Может использоваться как для запоминания, так и для выборки данных.

Инвертированный.

Значения ключей физических записей необязательно находятся в логической последовательности выборки данных каждого инвертируемого поля.

Эффективность доступа зависит от числа записей базы данных, числа уровней индексации и распределения памяти для размещения индекса.

Оценим эти два метода доступа внутренней модели: индексно-произвольный и инвертированный:

- При индексно-произвольном методе доступа записи хранятся в произвольном порядке. Создается отдельный файл из статей, содержащих значения действительного ключа и физические адреса хранимых записей.
- Как правило, инвертированный метод доступа применяется только для выборки. Запоминание осуществляется с помощью каких-либо других методов доступа.

Прямой метод доступа._Не требуется упорядоченность значений ключей физических записей. Между ключом записи и ее физическим адресом существует взаимно однозначное соответствие. Может применяться как для хранения, так и для поиска.

Эффективность доступа всегда равна единице.

Эффективность хранения зависит от плотности ключей. Наличие дубликатов ключей недопустимо.

Метод доступа посредством хеширования. Не требуется логическая упорядоченность значений ключей физических записей. Значениям нескольких ключей может соответствовать один и тот же физический адрес (блок). Может применяться как для хранения, так и для поиска.

Эффективность доступа зависит от распределения ключей, алгоритма их преобразования и распределения памяти.

Эффективность хранения зависит от распределения ключей и алгоритма их преобразования.

Между этими двумя методами доступа внутренней модели: прямым и посредством хеширования, существует сходство. При методе доступа посредством хеширования адрес физической записи алгоритмически определяется из значения ключа записи.

Разработчику базы данных приходится находить приемлемое соотношение между эффективностью доступа и эффективностью хранения.

Эффективность доступа приобретает первостепенное значение в оперативных системах. Некоторые СУБД сконструированы так, чтобы обновления (базы данных) в оперативном режиме выполнялись оптимальным образом. Эффективность доступа косвенно зависит и от эффективности хранения.

С помощью рассмотренных выше шести методов доступа внутренней модели можно реализовать другие, более сложные методы доступа.

6.9.2. Методы доступа внешней модели (представления пользователя)

Методы доступа, описывающие логические взаимосвязи, мы называем методами доступа внешней модели. На основе модели данных, используемой СУБД (например: реляционной, иерархической, сетевой или какого-либо их сочетания), могут быть получены различные представления пользователя (внешние модели), описывающие взаимосвязь (взаимосвязи) между различными частями физической записи.

Методы доступа внутренней модели обеспечивают вхождение в базу данных, а методы внешней модели выполняют дальнейший поиск записей базы данных или их частей с помощью взаимосвязей между записями. Методы доступа внешней модели осуществляют их хранение или поиск взаимосвязи между записями, определяются моделью данных, используемой конкретной СУБД в качестве основной структуры данных. Для отыскания

записи можно воспользоваться несколькими методами доступа внешней модели, но для хранения - только одним.

ПРАКТИЧЕСКИЕ ЗАДАНИЯ

Практическое задание 6.1

Постановка задачи

В качестве основной задачи, на которой будем изучать методологии и технологии проектирования и реализации баз данных, рассмотрим создание базы данных «Учет отпуска готовой продукции со склада предприятия «Метиз-М»». Готовая продукция со склада отпускается кладовщиком при наличии у покупателя документа «Требование». Документ выписывается сотрудником отдела продаж при наличии товара на складе и при условии произведенной оплаты по запросу покупателя в случае свободной продажи или в рамках заключенного договора на производство необходимых крепежных изделий. При этом покупателю выдаются сопроводительные документы на товар, производятся соответствующие изменения информации о состоянии склада, передается сообщение менеджеру по договорам для закрытия договора, выдаются ежедневные и ежемесячные отчеты о продажах.

Выполнение работы

1. Ознакомиться с предложенным вариантом описания предметной области. Проанализировать предметную область, уточнив и дополнив ее, руководствуясь собственным опытом, консультациями и другими источниками.
2. Определить основные допущения при описании предметной области.
3. Выполнить словесное описание работы подразделения, алгоритмов и сценариев выполнения отдельных работ.
4. Построить диаграммы потоков данных, отражающие логику обработки данных.

Практическое задание 6.2

Для определения объектов ПО и их характеристик проведите анализ ПО.

1. Рассмотрите хранилища данных и их составные элементы.
2. Составьте таблицу соответствия.
3. Выделите объекты и их характеристики.
4. Проведите нормализацию.
5. Постройте концептуальную и логическую модели данных.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

1. Дайте характеристику понятия проектирование БД.
2. Какие этапы включает в себя процесс проектирования базы данных?
3. Каким требованиям должен отвечать проект базы данных?
4. Каковы задачи, решаемые на этапе концептуального проектирования?
5. Какие вы знаете подходы к концептуальному проектированию?
6. В чем состоит отличие понятия типа сущности и элемента сущности?
7. Каковы способы представления сущности?
8. Каковы правила атрибутов?
9. Как классифицируются атрибуты?
10. Каковы фундаментальные виды связей?

11. Как формализуется связь 1:1?

12. Как формализуется связь 1:M?

13. Как формализуется связь M:N?

14. Что такое подтип и супертип?

15. Суть процесса определения требований к операционной обстановке, на каком этапе проектирования осуществляется?

16. На что следует ориентироваться при выборе СУБД и инструментальных программных средств?

17. Каковы задачи, решаемые на этапе логического проектирования?

18. Каковы задачи, решаемые на этапе физического проектирования?

19. Почему на современном этапе развития технологий БД проектировщика часто не устраивает проектирование реляционной базы данных в терминах отношений?

20. Дайте характеристику метода нормализации.

21. Смысл нормальных форм ER-схем.

22. Каковы этапы получения реляционной схемы из ER-схемы?

23. Какие вам известны наиболее популярные средства проектирования данных?

24. Каковы функции АБД при проектировании баз данных?

25. Назначение и формы реализации Словаря данных.

26. Требования и организация идеального Словаря данных.

27. Способы сбора и анализа информации при создании концептуальной модели.

28. В чем суть процесса создания логической модели?

29. Суть процесса создания физической модели.

30. Что такое DDL -скрипт для создания объектов базы данных?

31. Из каких шагов состоит процесс физической реализации базы данных?

32. Какие методы хранения и доступа к данным вам известны?

ГЛАВА 7. ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИЙ WWW ДЛЯ ДОСТУПА К БАЗАМ ДАННЫХ

Многие организации используют электронные базы данных (БД) для поддержки своих рабочих процессов. Часто это системы на одного - двух пользователей, выполненные с использованием dbf - ориентированных средств разработки: Clipper, Dbase, FoxPro, Paradox, Access. Обычно используется ряд таких баз, независимых друг от друга. Если информация, хранимая в таких БД, представляет интерес не только для непосредственных пользователей, то для ее дальнейшего распространения используются бумажные отчеты и справки, созданные базой данных.

С появлением локальных сетей, подключением таких сетей к Интернет, созданием внутрикорпоративных, сетей, появляется возможность с любого рабочего места организации получить доступ к информационному ресурсу сети. Однако, при попытке использовать существующие БД возникают проблемы связанные с требованием к однородности рабочих мест (для запуска "родных" интерфейсов), сильнейшим трафиком в сети (доступ идет напрямую к файлам БД), загрузкой файлового сервера и невозможностью удаленной работы (например, командированных сотрудников). Решением проблемы могло бы стать использование унифицированного интерфейса WWW для доступа к ресурсам организации.

Технология World Wide Web, в переводе "Всемирная паутина", получила столь широкое распространение из-за простоты своих пользовательских интерфейсов. Принцип "жми на то, что интересно", лежащий в основе гипертекста, интуитивно понятен. В технологиях WWW все ключевые понятия просматриваемого документа: слова, картинки - имеют возможность "раскрыться" новым документом, развивающим это понятие. Такой способ представления информации называется "гипертекстом", а документы, представленные в таком виде - "гипертекстовыми документами". Для описания этих документов используется специальный язык - язык описания гипертекстовых документов или HTML (англ. вариант HyperText Markup Language).

Из этих предпосылок возникает задача преобразования накопленных данных в гипертекстовые документы WWW, задача поддержки актуальности преобразованной структуры. Другими словами, задача предоставления WWW - доступа к существующим базам данных.

7.1. Основные понятия

Использование технологий WWW для обеспечения доступа к каким-либо информационным ресурсам подразумевает существование следующих компонент (рис.7.1):

1. IP - сети с поддержкой базового набора услуг по передаче данных с единой политикой нумерации и маршрутизации, работающим сервисом имен DNS.
2. Выделенного информационного сервера - WWW-сервера, обеспечивающего предоставление гипертекстовых документов через IP - сеть в ответ на запросы WWW - клиентов.

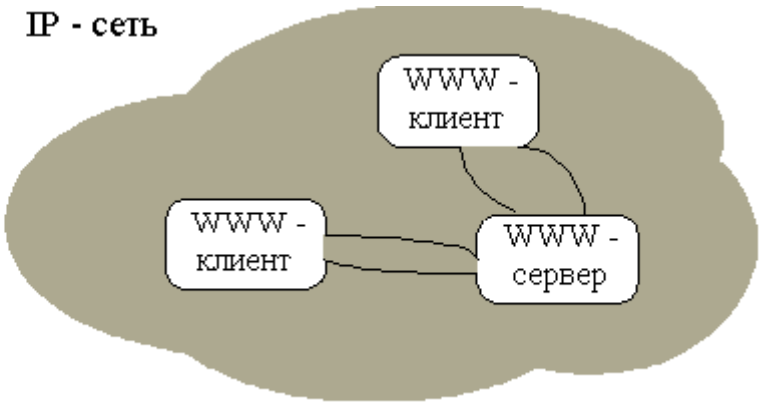

Рис. 7.1. Компоненты WWW технологий

Передаваемые гипертекстовые документы оформляются в стандарте HTML - языке описания гипертекстовых документов. Эти документы могут либо храниться в статическом виде (совокупность файлов на диске), либо динамически компоноваться в зависимости от параметров запроса специальным программным обеспечением. Для динамической компоновки HTML-документов, WWW-сервер использует специальным образом оформленные программы - CGI-программы.

7.2. Сценарии WWW - доступа к существующим базам данных

В состав специфики конкретной БД входят как технологические основы, такие как тип СУБД, вид интерфейсов, связи между таблицами, ограничения целостности, так и организационные решения, связанные с поддержкой актуальности баз данных и обеспечением доступа к ней.

При обеспечении WWW-доступа к существующим БД, возможен ряд путей - комплексов технологических и организационных решений. Практика использования WWW-технологии для доступа к существующим БД предоставляет широкий спектр технологических решений, по-разному связанных между собой - перекрывающих, взаимодействующих и т.д. Выбор конкретных решений при обеспечении доступа зависит от специфики конкретной СУБД и от ряда других факторов, как то: наличие специалистов, способных с минимальными издержками освоить определенную ветвь технологических решений, существование других БД, WWW-доступ к которым должен осуществляться с минимальными дополнительными затратами и т.д.

WWW - доступ к существующим базам данных может осуществляться по одному из трех основных сценариев. Ниже дается их краткое описание и основные характеристики.

Сценарий 1. Однократное или периодическое преобразование содержимого БД в статические документы.

В этом варианте содержимое БД просматривает специальная программа, создающая множество файлов - связных HTML-документов (рис.7.2). Полученные файлы могут быть перенесены на один или несколько WWW-серверов. Доступ к ним будет осуществляться как к статическим гипертекстовым документам сервера.

Этот вариант характеризуется минимальными начальными расходами. Он эффективен на небольших массивах данных простой структуры и редким обновлением, а также при пониженных требованиях к актуальности данных, предоставляемых через WWW. Кроме этого, очевидно полное отсутствие механизма поиска, хотя возможно развитое индексирование.

В качестве преобразователя может выступать программный комплекс, автоматически или полуавтоматически генерирующий статические документы. Программа-преобразователь может являться самостоятельно разработанной программой либо быть интегрированным средством класса генераторов отчетов.

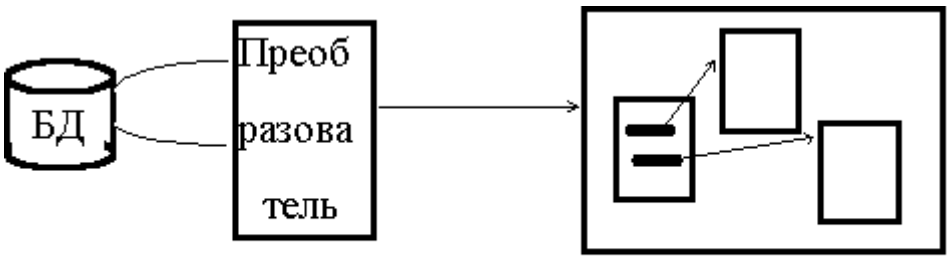

Рис. 7.2. Реализация сценария 1

Сценарий 2. Динамическое создание гипертекстовых документов на основе содержимого БД.

В этом варианте доступ к БД осуществляется специальной CGI-программой, запускаемой WWW-сервером в ответ на запрос WWW - клиента. Эта программа, обрабатывая запрос, просматривает содержимое БД и создает выходной HTML-документ, возвращаемый клиенту (рис.7.3).

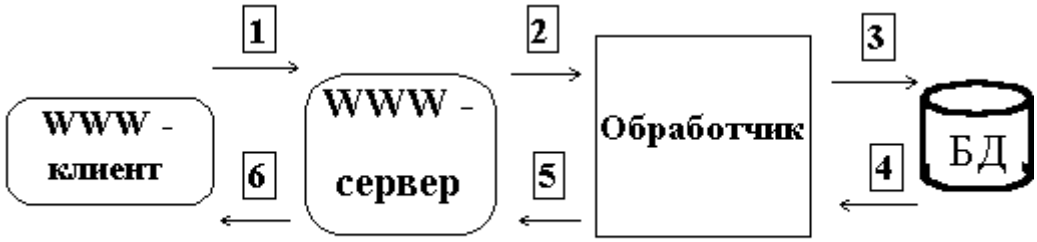

Рис. 7.3. Реализация сценария 2

Это решение эффективно для больших баз данных со сложной структурой и при необходимости поддержки операций поиска. Показаниями также являются частое обновление и невозможность синхронизации преобразования БД в статические документы с обновлением содержимого. В этом варианте возможно осуществлять изменение БД из WWW-интерфейсов.

К недостаткам этого метода можно отнести большое время обработки запросов, необходимость постоянного доступа к основной базе данных, дополнительную загрузку средств поддержки БД, связанную с обработкой запросов от WWW - сервера.

Для реализации такой технологии необходимо использовать взаимодействие WWW-сервера с запускаемыми программами CGI - Common Gateway Interface. Выбор программных средств достаточно широк - языки программирования, интегрированные средства типа генераторов отчетов. Для СУБД с внутренними языками программирования существуют варианты использования этого языка для генерации документов.

Сценарий 3. Создание информационного хранилища на основе высокопроизводительной СУБД с языком запросов SQL. Периодическая загрузка данных в хранилище из основных СУБД.

В этом варианте предлагается использование технологии, получившей название "информационного хранилища" (ИХ). Для обработки разнообразных запросов, в том числе и от WWW-сервера, используется промежуточная БД высокой производительности (рис.7.4). Информационное наполнение промежуточной БД осуществляется специализированным программным обеспечением на основе содержимого основных баз данных (рис.7.5).

Данный вариант свободен ото всех недостатков предыдущей схемы. Более того, после установления синхронизации данных информационного хранилища с основными БД возможен перенос пользовательских интерфейсов на информационное хранилище, что существенно повысит надежность и производительность, позволит организовать распределенные рабочие места.

Несмотря на кажущуюся громоздкость такой схемы, для задач обеспечения WWW-доступа к содержимому нескольких баз данных накладные расходы существенно уменьшаются.

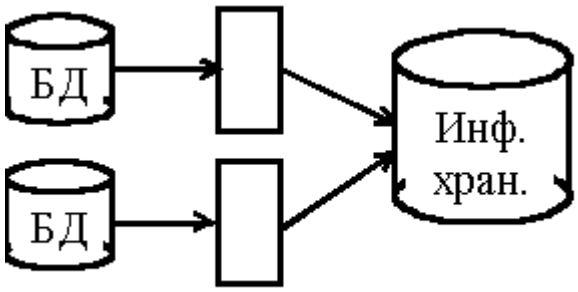

Рис. 7.4. Этап 1 - перегрузка данных

Основой повышения производительности обработки WWW-запросов и резкого увеличения скорости разработки WWW-интерфейсов является использование внутренних языков СУБД информационного хранилища для создания гипертекстовых документов.

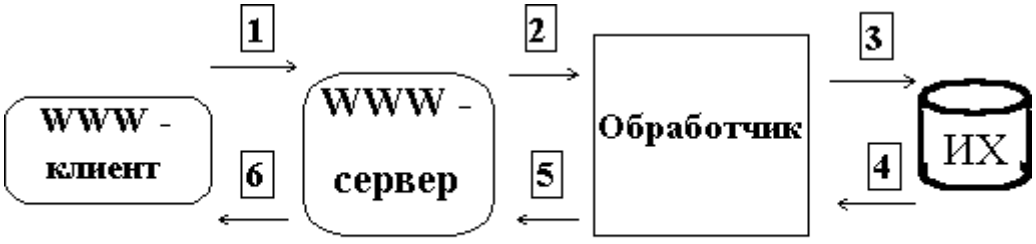

Рис.7.5. Этап 2 - обработка запросов

Для загрузки содержимого основной БД в информационное хранилище могут использоваться все перечисленные решения (языки программирования, интегрированные средства), а также специализированные средства перегрузки, поставляемые с SQL-сервером и продукты поддержки информационных хранилищ.

7.3. Обзор технологий

WWW - сервер NCSA HTTPD

Как было сказано ранее, одним из ключевых элементов технологии WWW является WWW-сервер. Стандартом де-факто для Unix-систем стало программное обеспечение (ПО) WWW-сервера Национального Центра по Суперкомпьютерным Приложениям (NCSA) Иллинойского Университета. Все вновь создаваемые продукты поддерживают полную совместимость с ПО NCSA по режимам работы и форматам данных. Сервер NCSA является постоянно совершенствуемым продуктом, отражающим последние веяния WWW-технологии. Созданная относительно недавно "Apache Group" разрабатывает свое программное обеспечение WWW - сервера на базе продукта NCSA HTTPD.

SQL - сервер фирмы Oracle

При реализации сценария 3 встает вопрос о выборе качественной платформы для создания информационного хранилища. Реляционная система управления базами данных фирмы Oracle является лидером на рынке СУБД. По производительности, надежности хранения данных, развитию семейства интерфейсов, объему серверных платформ продукты Oracle возглавляют многочисленные рейтинги. Гибкость использования, развитые средства управления доступом и распределенная архитектура делают сервер Oracle чрезвычайно привлекательным для технологии информационных хранилищ, а возможность работы на свободно - распространяемых Unix-платформах расширяет его возможности в некоммерческой среде.

Существенным ограничением использования Oracle в сфере науки и образования является достаточно высокая цена и низкое бюджетное финансирование. Однако с 1996 года фирма Oracle объявила о специальной программе для российских университетов, что позволяет за относительно небольшие деньги приобрести любой набор продуктов Oracle.

Библиотеки и функции на языке C

Одной из основных технологий создания CGI-модулей для реализации функций "преобразователя" и "обработчика" сценариев 1-3 является язык C. Язык C - наиболее распространенный язык программирования. В каждом ВУЗе есть специалисты, способные использовать его для создания приложений. При решении описанных задач язык C можно использовать для создания следующих программ:

1. преобразователя, однократно преобразующего содержимое БД в сеть гипертекстовых документов (рис. 7.2);
2. обработчика, динамически обрабатывающего запрос от WWW-сервера к БД (рис. 7.3);
3. перегрузчика из существующих БД в информационное хранилище (рис. 7.4);
4. обработчика запросов от WWW-сервера к информационному хранилищу (рис. 7.5).

Для поддержки этих функций создано большое количество библиотек и функций языка C, готовых приложений в исходных текстах.

Пакет Web - Oracle - Web

Пакет WOW является свободно-распространяемым программным средством, предназначенным для создания интерактивных WWW-интерфейсов с СУБД Oracle. Пакет WOW был первым и наиболее простым средством, выпущенным фирмой Oracle. В настоящее время существует набор продуктов, развивающих функциональность WOW'a - Oracle Web Server версий 1, 2, Oracle Web Arcitecture.

Все перечисленные продукты позволяют использовать процедурное расширение языка SQL - PL/SQL, разработанное фирмой Oracle для динамического создания гипертекстовых

документов. Высокая скорость разработки достигается за счет резкого упрощения доступа к БД - программы на PL/SQL исполняются самим сервером Oracle. Предлагаемый пакет WOW был переработан в Новосибирском областном центре НИТ с целью поддержки нескольких русскоязычных кодировок.

Основной областью использования WOW является обработка запросов от WWW-сервера к SQL-серверу Oracle в среде Unix. В предложенных сценариях пакет WOW позволит организовать эффективный WWW доступ к информационному хранилищу, построенному на базе сервера баз данных Oracle (сценарий 3).

Пакет Cold Fusion фирмы Allaire Corp

Пакет предназначен для использования под ОС Windows и позволяет обращаться к различным базам данных, поддерживающим интерфейс ODBC через WWW-интерфейсы. Пакет имеет коммерческий статус, его "evaluation copy" является свободно-распространяемой. Для доступа к базам данных используются конструкции языка DBML - расширения языка HTML, дополненного средствами доступа к БД через ODBC. Документы на языке DBML обрабатываются на серверной части, в результате чего создается HTML-документ. Полноценная версия пакета, вместе с WWW - сервером стоит $486.

Пакет может эффективно использоваться в качестве обработчика запросов WWW к исходным базам данных или информационному хранилищу (сценарии 2,3).

7.4. Оценка трудоемкости обеспечения WWW доступа

Трудоемкость обеспечения WWW-доступа к базам данных, очевидно, складывается из трудоемкости работ при реализации одного из вышеприведенных сценариев. Реализация первого сценария связана с последовательным преобразованием всех данных, находящихся в исходной БД. Разработка средств вывода содержимого таблицы в формате HTML с необходимым форматированием и текстовым сопровождением будет занимать порядка 1-3-х дней для одного разработчика. Разработка средств построения индексной структуры к выводимым данным является более творческой работой и может занять 1-3 недели для одного разработчика.

Трудоемкость построения интерфейсов для сценариев 2, 3, в общем случае, эквивалентна трудоемкости построения этих интерфейсов при создании исходной информационной системы (т.е. той, для которой обеспечивается WWW-доступ) с использованием традиционных средств разработки (не-CASE). В третьем сценарии дополнительные трудозатраты пойдут на перегрузку данных в ИХ. При перегрузке данных без изменения структуры и имен можно исходить из оценки трудозатрат: 1-2 таблицы в 1-2 дня для одного разработчика, в зависимости от сложности и объема таблиц, при условии отладки технологии перегрузки.

При использовании различных средств разработки интерфейсов к БД трудозатраты могут существенно различаться. Ранжированный по уменьшению трудозатрат на разработку интерфейсов список будет выглядеть так:

1. библиотеки и функции на языке C;
2. язык Perl;
3. - 4. пакеты WOW и Cold Fusion.

7.5. Информационные системы на основе Internet/Intranet-технологии

Изначально технология Internet/Intranet/WWW предназначалась для облегчения доступа к информации и публикации документов. Программа-клиент выполняет функции интерфейса пользователя и обеспечивает доступ практически ко всем информационным ресурсам Internet.

База данных HTML-документов - это часть файловой системы, которая содержит текстовые файлы в формате гипертекста и связанные с ними графику и другие ресурсы. Фактически браузер является интерпретатором HTML-текста. И, как типичный интерпретатор, клиент в зависимости от команд разметки выполняет различные функции. В круг этих функций входит не только размещение текста на экране, но и обмен информацией с сервером по мере анализа полученного HTML-текста, что наиболее наглядно происходит при отображении встроенных графических образов.

При анализе URL-спецификаций или по командам сервера клиент запускает дополнительные внешние программы для работы с документами в форматах, отличных от HTML, например GIF, JPEG, MPEG, Postscript и т. п. В последнее время все большее распространение получает механизм согласования запускаемых программ через MIME-типы.

До недавнего времени сеть Internet была "улицей с односторонним движением" - информация с Web-страниц поступала к пользователю от Web-сервера при наличии запроса. Обратная связь с пользователем - анкетные листы и параметры запроса для работы поисковых систем. При этом программирование серверной части осуществляется с использованием CGI-скриптов.

Если Web-сервер использует традиционные статичные Web-страницы, то в ответ на запрос клиента Web-сервер передает страницу в формате HTML. Однако при работе с приложениями базы данных адрес URL указывает не на Web-страницу, а на программу или сценарий, который запускает запрос к базе данных и преобразует результаты в формат HTML. Затем Web-сервер посылает полученную HTML-страницу Web-клиенту. Так как этот процесс основан на технологии Web, клиентской платформой может стать любой компьютер, на котором исполняется Web-браузер, а серверной платформой - любая ЭВМ под управлением Web-сервера.

Использование CGI-скриптов имеет ряд недостатков - статичное представление информации, преобразование результата запроса - отчета в HTML-файл, отсутствие динамического просмотра изменения информации в базе данных. Кроме того, такой принцип работы перегружает каналы связи.

Предложенная фирмой Sun технология Java ориентирует взаимодействие между клиентом и сервером на поток команд, а не данных. В ходе сеанса обеспечивается фоновая подкачка через сеть на компьютер клиента программных агентов - апплетов, которые берут на себя функции обеспечения гибкого взаимодействия. Все, что нужно для этого, - встроить в Web-браузер исполняющую систему для апплетов.

7.6. Подготовка гипертекстовых документов для World Wide Web

Web-страницы описываются на специальном языке, называемом HTML (HyperText Markup Language, Язык разметки гипертекстовой информации), ставшем основным языком описания документов в Internet. HTML является простым подмножеством универсального языка разметки документов SGML (Standard Generalized Markup Language, Стандартный язык разметки документов), являющегося стандартом для обмена документами между различными платформами. Точнее, весь синтаксис HTML полностью описывается с

помощью SGML DTD (Document Type Definition). По этой причине почти все программы, совместимые с SGML, могут быть использованы при подготовке HTML-документов.

Интересно отметить некоторые особенности, отличающие верстку информации для Web и верстку для ``обычной'', то есть, бумажной технологии передачи документов. В отличие от языков описания печатных документов, вроде известного языка PostScript, упор делается на переносимость информационного наполнения страниц, а не их внешнего оформления. Поясним сказанное на примере: при переносе документа на языке PostScript между двумя компьютерами гарантируется сохранение его внешнего вида, то есть размеров, шрифтового оформления; тогда как для HTML-документов гарантируется лишь сохранение логической структуры. Это происходит потому, что никто не гарантирует, что устройство, на котором пользователь будет просматривать Web-страницу, не окажется черно-белым алфавитно-цифровым терминалом 1970-го года выпуска! Или же что программа просмотра, используемая пользователем, способна корректно отобразить графические вставки в различных форматах. И поэтому Web-дизайнер несет особую ответственность за представление информации на своих страницах.

7.7. Назначение WWW - сервера. Общая схема работы. Определение

WWW сервер - это такая часть глобальной или внутрикорпоративной сети, которая дает возможность пользователям сети получать доступ к гипертекстовым документам, расположенным на данном сервере. Для взаимодействия с WWW сервером пользователь сети должен использовать специализированное программное обеспечение - броузер (от англ. browser), другое название - программа просмотра.

Схема работы WWW-сервера. В общем виде она выглядит так:

1. Пользователь сети запускает пакет программного обеспечения, называемый броузером, в функции которого входит.
   - Установление связи с сервером
   - Получение требуемого документа
   - Отображение полученного документа
   - Реагирование на действия пользователя - доступ к новому документу
   - После запуска броузер по команде пользователя или автоматически устанавливает связь с заданным WWW - сервером и передает ему запрос на получение заданного документа (рис.7.6).
   -

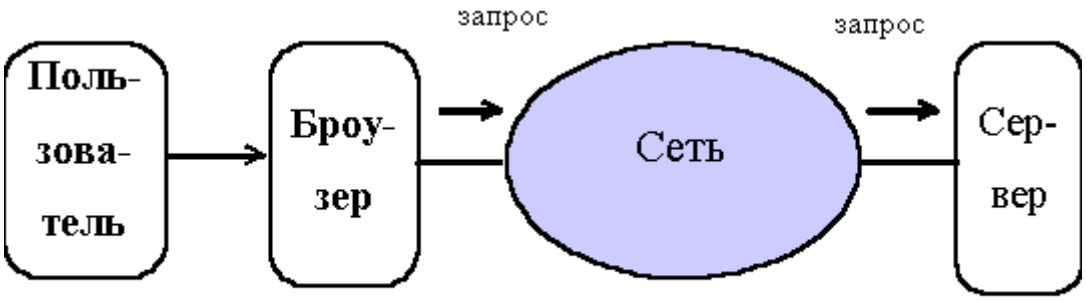

Рис.7.6. Этап 1 работы WWW – сервера

2. WWW - сервер ищет запрашиваемый документ и возвращает результаты броузеру (рис. 7.7).

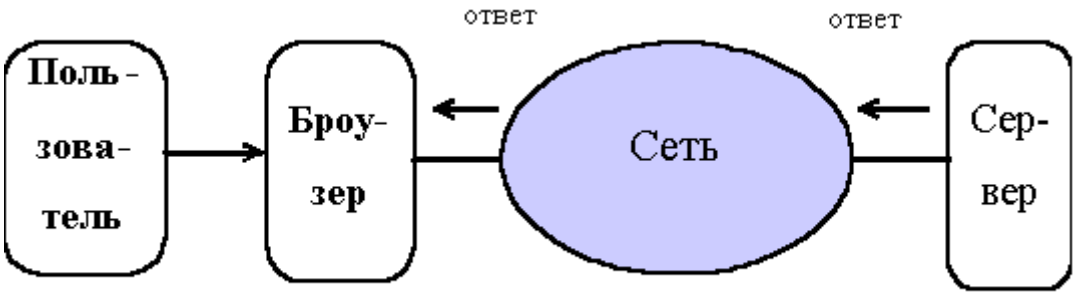

Рис.7.7. Этап 2 работы WWW – сервера

3. Броузер, получив документ, отображает его пользователю и ожидает его реакции. Возможные варианты:

1) Ввод адреса нового документа.

2) Печать, поиск, другие операции над текущим документом.

3) Активизация (нажатие) специальных зон полученного документа, называемых связями (link) и ассоциированными с адресом нового документа.

В первом и третьем случае происходит обращение за новым документом.

Адрес. Адрес документа указывается в виде специальной строки, называемой URL. Для протокола HTTP, используемого при взаимодействии WWW - клиента и WWW - сервера, URL состоит из следующих компонент:

- Наименование протокола, по которому работает сервер (http).
- Имя машины - сервера в Internet или ее IP - номер.
- Порт TCP, обращение к которому обрабатывает сервер.
- Место (путь) документа на машине - сервере.

Например:

http://www.cnit.nsu.ru:80/welcome.html

Здесь http означает протокол работы с WWW - сервером

':' - разделитель

"www.cnit.nsu.ru" - имя машины - сервера в Internet

"80" - номер tcp - порта

/welcome.html - путь до документа на машине - сервере

Из общей схемы работы видно, что функции WWW сервера заключаются в следующем:

- Установление соединения с клиентским ПО по протоколу tcp.
- Принятие запроса на документ по протоколу http.
- Поиск документа в локальных ресурсах.
- Возврат результатов поиска по протоколу http.

В общем случае, WWW - сервером будем называть программно - аппаратный комплекс, предназначенный для выполнения вышеперечисленных действий.

Среда работы сервера. В настоящее время все известные WWW - серверы представляют собой компьютер общего назначения с многозадачной операционной системой. Один или несколько процессов такой системы отвечают за поддержку специфических для WWW - сервера функций. Другие процессы ОС отвечают за обеспечение других функций, не обязательно связанных с поддержкой технологии WWW (рис. 7.8). Такая структура приводит к тому, что под WWW сервером начинают подразумевать только часть

программного обеспечения, единственными функциями которой являются функции WWW сервера, а остальную часть - компьютер, операционную систему, другие процессы, сетевую структуру называют средой работы WWW сервера или платформой.

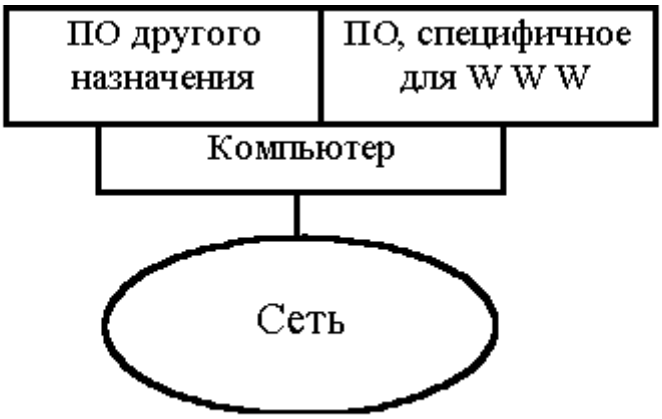

Рис. 7.8. Рабочая среда WWW – сервера

7.8. WWW и средства интерактивного взаимодействия

В простейшем случае гипертекстовый документ представляет собой совокупность файлов. Представление этих файлов как единого документа производится броузером. По каждому файлу документа броузер делает запрос к WWW - серверу. Таким образом, сервер не имеет представления о структуре и составе документов, он отвечает только за выдачу локальных файлов по запросам.

В общем случае, интерактивный интерфейс пользователя представляет собой систему, обеспечивающую взаимодействие пользователя и программы. Для WWW, интерактивный интерфейс можно определить как последовательность HTML-документов, реализующих интерфейс пользователя. Можно также условно классифицировать принципы построения интерфейса по типу формирования HTML-документа:

- статический
- динамический

В первом случае источником интерфейса является HTML-документ, созданный в каком-либо текстовом или HTML-ориентированном редакторе. Следовательно, данный документ остается неизменным в течение использования.

Во втором случае источником интерфейса является HTML-документ сгенерированный cgi-модулем. Следовательно, появляется некоторая гибкость в видоизменении интерфейса во время использования.

Таким образом, можно ввести понятие интерактивного интерфейса для WWW, который представляет собой последовательность статических или динамически формируемых HTML-документов, реализующих интерфейс пользователя.

Практически любая задача, решающая проблему получения данных от клиента, связана с построением интерфейса. Наиболее интересным является построение интерфейсов к различным базам данных, доступ к SQL-серверу, получение информации от периферийных устройств, создание клиентских рабочих мест. Все это возможно посредством CGI (Common Gateway Interface).

7.9. Интерфейс CGI

Помимо доступа к статическим документам сервера существует возможность получения документов как результата выполнения прикладной программы. Такая возможность реализуется на сервере WWW благодаря использованию интерфейса CGI (Common Gateway Interface). Спецификация CGI описывает формат и правила обмена данными между ПО WWW сервера и запускаемой программой.

Для инициирования CGI необходимо, чтобы в запрашиваемом URL был указан путь до запускаемой программы. ПО WWW сервера исполняет эту программу, передает ей входные параметры и возвращает результаты ее работы, как результат обработки запроса, клиенту. CGI - программой может являться любая программа локальной операционной системы сервера - в двоичном виде или в виде программы для интерпретатора (Basic, SH, Perl и т.д.).

С целью облегчения администрирования CGI - программ, а также для удовлетворения требованиям безопасности CGI - программы группируются в одном или нескольких явно указанных серверу каталогах. По умолчанию это каталог cgi-bin в иерархии серверных каталогов, однако, его имя и положение могут отличаться.

Например: клиент, обращающийся к CGI - программе test-query, будет использовать URL http://<имя_сервера>/cgi-bin/test-query

Интерфейс CGI позволяет расширить границы применения WWW - технологии. CGI - программа может обрабатывать сигналы с датчиков установок, взаимодействовать с мощным сервером баз данных, переводить и т.п. (см. рис. 7.9).

Common Gateway Interface (CGI) является стандартом интерфейса внешней прикладной программы с WWW сервером.

Задача построения вышеназванных интерфейсов делится на две части:

- Клиентская часть
- Серверная часть

Клиентская часть. Для создания клиентской части необходимо создать HTML-документ, в котором реализован интерфейс с пользователем. В языке HTML это возможно посредством форм.

Серверная часть. Серверная часть состоит из исполняемого модуля, решающего основные задачи обработки данных поступающих от клиентской части, формирования ответа в формате HTML, и т.д. Такой модуль называется cgi-модулем.

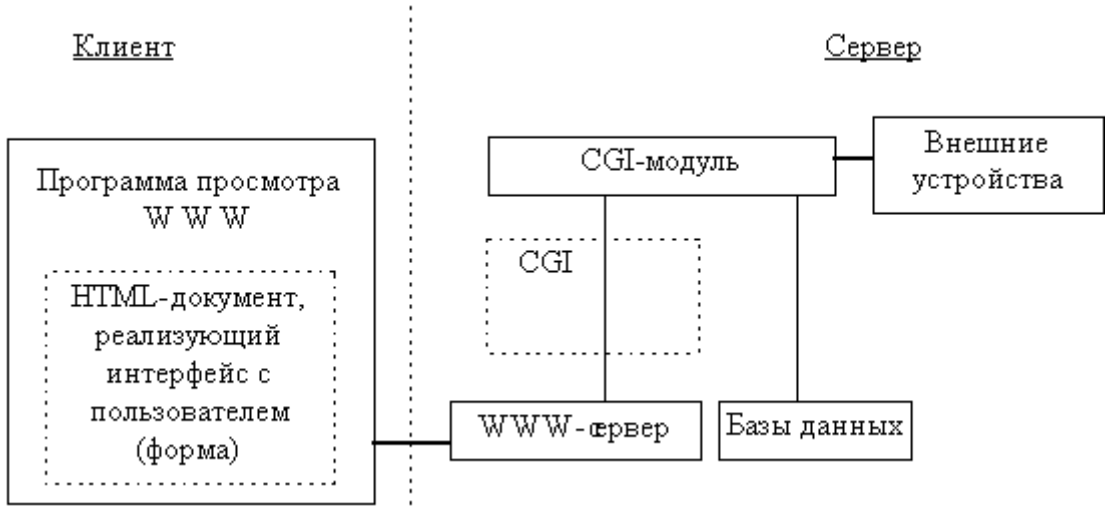

Рис. 7.9. Две части интерактивного интерфейса

Методы HTTP запроса. Для реализации взаимодействия "клиент-сервер" важно, какой метод HTTP запроса использует клиентская часть при обращении к WWW серверу. В общем случае, запрос - это сообщение, посылаемое клиентом серверу. Первая строка HTTP запроса включает в себя метод, который должен быть применен к запрашиваемому ресурсу, идентификатор ресурса(URI-Uniform Resource Identifier), и используемую версию HTTP-протокола. В рассматриваемом нами случае, клиентская часть применяет методы запроса POST и GET. Метод POST используется для запроса серверу, чтобы тот принял информацию, включенную в запрос, как относящуюся к ресурсу, указанному идентификатором ресурса. Метод GET используется для получения любой информации, идентифицированной идентификатором ресурса в HTTP запросе.

Для WWW-сервера стандарта NCSA прикладные программы или CGI-модули, обрабатывающие поток данных от клиента или (и) формирующие обратный поток данных могут быть написаны на таких языках программирования как:

- C/C++;
- Любой UNIX shell;
- Fortran;
- Perl;
- Visual Basic;
- TCL;
- AppleScript.

Спецификация CGI

CGI определяет 4 информационных потока (см. рис.7.10):

1. Переменные окружения
2. Стандартный входной поток
3. Стандартный выходной поток
4. Командная строка

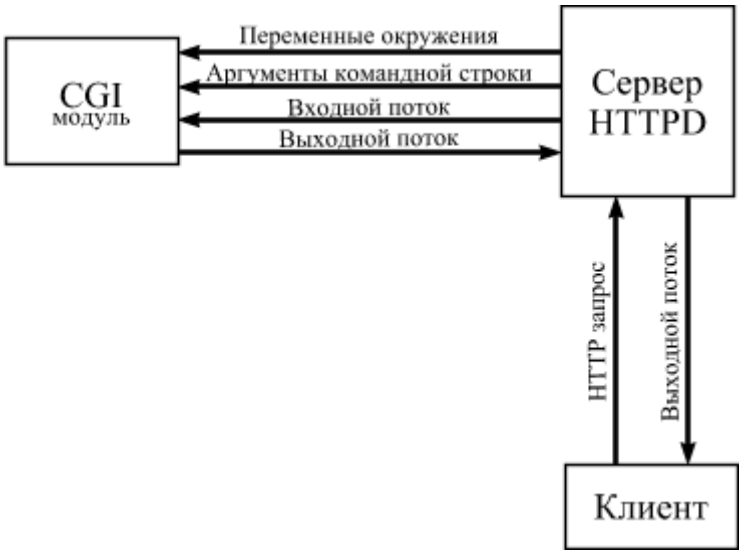

Рис.7.10. CGI-интерфейс

Переменные окружения. Переменные окружения условно делятся на два типа:

- общие для всех типов запросов (устанавливаются для всех типов),
- зависящие от метода запроса.

Стандартный вывод. CGI - модуль выводит информацию в стандартный выходной поток. Этот вывод может представлять собой или документ, сгенерированный cgi-модулем, или инструкцию серверу, где получить необходимый документ. Обычно cgi-модуль производит свой вывод. Преимущество такого подхода в том, что cgi-модуль не должен формировать полный HTTP заголовок на каждый запрос.

Заголовок выходного потока. В некоторых случаях необходимо избегать обработки сервером вывода cgi-модуля, и посылать клиенту данные без изменений. Для отличия таких cgi-модулей, CGI требует, чтобы их имена начинались на nph-. В этом случае формирование синтаксически правильного ответа клиенту cgi-модуль берет на себя.

ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

1. Дайте характеристику языка описания гипертекстовых документов.
2. Какие компоненты технологий WWW могут быть использованы для обеспечения доступа к каким-либо информационным ресурсам?
3. Дайте характеристику комплексу технологических и организационных решений по обеспечению WWW-доступа к существующим БД.
4. Отчего зависит выбор конкретных решений при обеспечении WWW-доступа к базам данных?
5. Каким образом может быть оценена трудоемкость обеспечения WWW-доступа к базам данных?
6. На что указывает адрес URL при работе с приложениями базы данных? Преимущества использования CGI-скриптов.
7. Недостатки использования CGI-скриптов.
8. Дайте определение WWW – сервера.
9. Для чего может быть использован WWW – сервер?
10. Дайте общую характеристику схемы работы WWW – сервера.
11. Из каких компонентов состоит среда работы WWW – сервера?
12. Что из себя представляет интерактивный интерфейс WWW?
13. Классифицируйте принципы построения интерфейса по типу формирования HTML-документа.
14. Дайте определение понятия интерактивного интерфейса для WWW.
15. Дайте определение Common Gateway Interface (CGI).
16. Что необходимо для инициирования CGI?
17. Каким образом осуществляется администрирование CGI – программ?
18. Как обеспечивается безопасность CGI – программы?
19. Из каких составных частей строится интерактивный интерфейс WWW?
20. Состав клиентской части интерактивного интерфейса WWW.
21. Состав серверной части интерактивного интерфейса WWW.
22. Дайте характеристику информационных потоков, которые определяет CGI.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ANSI/X3/SPARC Study Group on Data Base Management Systems Interim Report. FDT Bulletin, 7(2), 2005, pp. 1-140.
2. Barrie Sosinsky, IDG Books Worldwide, Inc. 2007
3. Ben-Or M. Lower bounds for algebraic computation trees // Proc. 15th ACM Annu. Symp. Theory Comput. – Apr. 2003. – P. 80-86.
4. Chazelle B.M. Filtering search: a new approach to query-answering // Proc. 24th IEEE Annu. Symp. Found. Comput. Sci. – Nov. 2003. – P. 122-132.
5. Chen P. The Entity- Relationship Approach to logical data base design. Database Management, 2006, №6.
6. Chen P.P. The Entity-Relationship Model: Toward a Unified View of Data. ACM Transactions on Database Systems, vol.1., № 1, 2006.
7. CODASYL – The Stored-Data Definition and Translation Task Group, Stored-Data Description and Data Translation: A Model and Language // Inf. Syst. – 1977. – V. 2, 3. – P. 95-148.
8. CODASYL Data Base Task Group Report. October 1969. and D.K. Barry. Morgan Kaufmann Publishers, Inc.,1997.
9. CODASYL Systems Committee, Feature Analysis of Generalized Data Base Management Systems, ASM, New York, 1971.
10. CODASYL Systems Committee, Selection and Acquisition Of Data Base Management Systems, ASM, New York, Mar. 1976.
11. Codd E.F. A Data Base Sublanguage Founded on the Relational Calculus // Proc. of the 1971 ACM-SIGFIDET Workshop on Data Description, Access, and Control, ACM, New York, London, Amsterdam, 1972.
12. Codd E.F. Access Control for Relational Data Base Systems // BCS Symposium on Relational Data-Base Concepts, Apr. 1973, British Computer Soc., London, 1973.
13. Codd E.F. Further Normalization of the Data Base Relational Model // Courant Computer Sci. Symposia (vol. 6: "Data-Base System"), ed. by R.Rustin, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1972.
14. Codd E.F. Recent Investigations in Relational Data-Base Systems // Information Processing'74, North-Holland, Amsterdam, 1974.
15. Codd E.F. Relational Completeness of Data Base Sublanguages // Courant Computer Sci. Symposia (vol. 6: "Data-Base System"), ed. by R.Rustin, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1972.
16. Codd E.F. Relational Database: A Practical Foundation for Productivity // Commun. of ACM. - 1982. – V. 25, 2. - P.140-155.
17. Codd E.F. Relational Model of Data for Large Shared Data Banks. Comm. Of the ACM, 1970, v.13, no.6, pp.377-387. (Есть русский перевод: Кодд Е.Ф. Реляционная модель для больших совместно используемых банков данных // Е.Ф. Кодд. СУБД. – 1995. - №1. – С.145-160.)
18. Data Language/I-System/370 DOS/VS, General Information Manual GH20-1246, IBM, White Plains, New York, 1974.
19. Date C.J., Codd E.F. The Relational and Network Approaches: Comparison of the Application Programming Interfaces // Proc. of the 1974 ACM-SIGFIDET Workshop, ACM, New York, London, Amsterdam, 1974.

20. Dobkin D.P. A nonlinear lower bound on search tree programs for solving knapsack problems // J. Comput. Syst. Sci. - 1976. – V. 13. – P. 69-73.

21. Dobkin D.P., Lipton R.J. On the complexsity of computations under varying sets of primitives // J. Comput. Syst. Sci. - 1979. – V. 18. – P. 86-91.

22. Edelsbrunner H., Overmars M.H., Siedel R. Some methods of computational geometry applied to computer graphics // IIG, Technische Univ. Graz, Austrua, Tech. Rep. F117. – June 1983.

23. Gasanov E.E. Instantly solvable search problems // Proceedings of International Symposium on Intelligent Data Analysis (IDA-95). Baden-Baden, Germany. – IIAS Press, 1995. - P. 65-69.

24. Gasanov E.E. On fast solving of interval search problem // Proceedings of International Congress of Mathematicians. Zurich, Switzerland, 1994. – P. 137.

25. Information Management System Virtual Storage (IMS/VS), General Information Manual GH20-1260, IBM, White Plains, New York, 1974.

26. Information Management System/360, Version 2, Application Programming Reference Manual SH20-0912, IBM, White Plains, New York, 1974.

27. ISO/IEC 9075- 2:1999, Information technology – Database language – SQL – Part 2: Foundation (SQL/ Foundation), 1999. - 1121 p.

28. ISO/IEC 9075- 3:1999, Information technology – Database language – SQL – Part 3: Call-Level Interface(SQL/ CLI), 1999. - 401p.

29. ISO/IEC 9075- 4:1999, Information technology – Database language – SQL – Part 4: Persistent Stores Modules (SQL/ PSM), 1999. - 152 p.

30. ISO/IEC 9579:2000/ Information technology – Remote Database Access for SQL (RDA/SQL) ISO/IEC 9075- 2:1999, Information technology – Database language – SQL – Part 2: Foundation (SQL/ Foundation), 1999. - 1121 p.

31. Lee D.T., Wong C.K. Quintari trees: A file structures for multidimensional database system // ACM Trans. Database Syst. - Sept. 1980. – V. 1, 1. – P. 339-353.

32. Proc. of Conference on Data: Abstraction, Definition and Structure. SIGPLAN Notices, 11, 1976.

33. Steele J.M., Yao A.C. Lower bounds for algebraic decision trees // J. Algorith. – 1982. – V. 3. – P. 1-8.

34. The Object Database Standard ODMG-93. Edited by R.G.G. Cattell. Morgan Kaufmann Publishers, 1994.

35. The Object Database Standard: ODMG 2.0,-93. Edited by R.G.G. Cattell and D.K. Barry. Morgan Kaufmann Publishers, Inc., 1997.

36. The Object Database Standard: ODMG 3.0. Edited by R.G.G. Cattell and D.K. Barry. Morgan Kaufmann Publishers, Inc., 2000.

37. Yao A.C., Rivest R.L. On the polyhedral decision problem // SIAM J. Comput. – 1980. – V. 9. – P. 343-347.

38. Абчук В.А. Поиск объектов./ В.А. Абчук, В.Г. Суздаль - М.: Советское радио, 2007.

39. Альсведе Р. Задачи поиска./ Р. Альсведе, И. Вегенер - М.: Мир, 2002.

40. Атре Ш. Структурный подход к организации баз данных./ Ш. Атре - М.: Финансы и статистика, 1983.

41. Ахо А. Построение и анализ вычислительных алгоритмов./ А. Ахо, Дж. Хопкрофт, Дж. Ульман - М.: Мир, 2009.

42. Баргесян А. А. Технологии анализа данных: Data Mining, Visual Mining, Text Mining, OLAP / А. А. Баргесян, Куприянов М.С., Степаненко В.В., Холод И.И. – Спб. : BHV-СПб, 2008. – 384 с.

43. Бойко В.В. Проектирование баз данных информационных систем./ В.В. Бойко, В.М Савинков. – М., Финансы и статистика, 2009.

44. Васкевич Д. Стратегии клиент/сервер: Руководство по выживанию для специалистов по реорганизации бизнеса. / Д. Васкевич – Киев: Диалектика, 2006. – 384 с.

45. Вендров А.М. CASE – технологии. Сравнительный анализ и классификация./ А.М. Вендров, В.П. Бобков – Сборник научных трудов, М.: МЭСИ, 2004.

46. Вендров А.М. CASE-технологии. Современные методы и средства проектирования информационных систем./ А.М. Вендров – М.: Финансы и статистика, 2008.

47. Вендров А.М. Проектирование программного обеспечения экономических информационных систем: Учебник./ А.М. Вендров – М.: Финансы и статистика, 2000.– 352 с.

48. Гасанов Э.Э. Математические модели и сложность информационного поиска // Proceedings of the graduate workshop in mathematics and its applications in social sciences. Ljubljana. / Э.Э. Гасанов – 1991. – P. 37-53.

49. Гасанов Э.Э. О сложности поиска в базах данных // Искусственный интеллект: Межвузовский сборник трудов./ Э.Э. Гасанов - Саратов, Изд-во Саратовского университета, 1993. – С. 41-56.

50. Горев А. Эффективная работа с СУБД./ А. Горев, Р. Ахаян, С. Макашарипов. - изд. "Питер",2007

51. Горин С.В. CASE-средство S-Designor 4.2 для разработки структуры базы данных./ С.В. Горин, А.Ю. Тандоев // СУБД, N 3, 1995.

52. Горин С.В. Применение CASE-средства ERwin 2.1 для информационного моделирования в системах обработки данных./ С.В. Горин, А.Ю. Тандоев // СУБД, N 3, 1995.

53. ГОСТ 34.320-96 Информационная технология. Система стандартов по базам данных. Концепции и терминология для концептуальной схемы и информационной базы.

54. ГОСТ 34.321- 96 Информационная технология. Система стандартов по базам данных. Эталонная модель.

55. ГОСТ Р ИСО/МЭК 12207-99 Информационная технология. Процессы жизненного цикла программных средств.

56. Грофф Джеймс Р. SQL: Полное руководство: Пер. с англ./ Грофф Джеймс Р., Вайнберг Пол Н. – Киев: BHV, 2004. – 608 с.

57. Гуда А.Н., Бутакова М.А. и др. Информатика. Общий курс: Учебник/ Под ред. академика РАН В.И. Колесникова- М.: Дашков и К, 2008.- 400 с.- Допущено УМО.

58. Данные в языках программирования. Абстракция и типология: Сб. статей; Пер. с англ./ Под ред. В.Н.Агафонова. – М.:Мир, 2002.

59. Дейт К.Дж. Введение в системы баз данных 8-е издание: Пер. с англ./ Дейт К.Дж. - К.;М.;СПб.: Издательский дом «Вильямс»,2006. – 848 с.

60. Дж. Ульман. Основы систем баз данных./ Дж. Ульман. – М.: Финансы и статистика, 1983.

61. Джексон Г. Проектирование реляционных баз данных для использования с микроЭВМ: Пер. с англ./ Г. Джексон – М.: Мир, 1991.

62. Диго С.М. Проектирование баз данных. / С.М. Диго – М.: Финансы и статистика, 1988.

63. Диго С.М. Проектирование и использование баз данных: Учебник для вузов./ С.М. Диго – М.: Финансы и статистика, 1995. – 208 с.

64. Ершов А.П. Некоторые субъективные замечания к актуальным проблемам программирования / А.П. Ершов // Перспективы системного и теоретического программирования: Труды Всесоюзного симпозиума, Новосибирск, 20-22 марта, 2004 г. - ВЦ СО АН РФ, Новосибирск, 2004. – С. 113-127.

65. Информационные системы общего назначения. М.: Статистика, 2001.

66. Ипатова Э.Р. Информационные системы: Учебное пособие./ Э.Р. Ипатова – Магнитогорск: МаГУ,2001. – 160 с.

67. Калянов Г.Н. CASE – структурный системный анализ (автоматизация и применение)./ Г.Н. Калянов – М.: Лори, 1996. - 242 с.

68. Калянов Г.Н. CASE – технологии./ Г.Н. Калянов // Hard & Soft, №7, 1995.

69. Калянов Г.Н. CASE: компьютерное проектирование программного обеспечения./ Г.Н. Калянов - М.: НИВЦ МГУ, 1994.

70. Калянов Г.Н. Методы и средства системного и структурного анализа и проектирования./ Г.Н. Калянов - М.: НИВЦ МГУ, 1996.

71. Калянов Г.Н. Системное проектирование – новый вид деятельности на российском рынке./ Г.Н. Калянов // Информационные технологии, № 3-4, 1995.

72. Калянов Г.Н. Современные CASE–технологии. / Г.Н. Калянов – М.: ИПУ, 2002.–250 с.

73. Кватрани, Т. Rational Rose 2000 и UML. Визуальное моделирование: Пер. с англ./ Т.Кватрани. – М.: ДМК Пресс, 2001.

74. Когаловский М.Р. XML: возможности и перспективы. Ч. 1. Платформа XML и составляющие ее стандарты / М.Р. Когаловский // Директор информационной службы. – Январь 2001.-С. 24-28.

75. Когаловский М.Р. Абстракции и модели в системах баз данных / М.Р. Когаловский // СУБД. -2001.- №4-5.- С. 73-81.

76. Когаловский М.Р. Проблемы терминологии в теории систем баз данных / М.Р. Когаловский // УСиМ. – 1986.-№6.-С. 85-92.

77. Козлов В.А. Открытые информационные системы / В.А.Козлов. – М.: Финансы и статистика, 1999. – 224 с.

78. Конноли Т. Базы данных: проектирование, реализация и сопровождение. Теория и практика, 2-е изд.: Пер. с англ.: Уч. пос./ Т. Конноли, К. Бегг, А. Страчан - М.: Издательский дом «Вильямс», 2000. – 1120 с.

79. Корнеев В.В. Базы данных. Интеллектуальная обработка информации. / В.В. Корнеев, А.Ф. Гареев, С.В. Васютин, В.В. Райх – М.: «Нолидж», 2000. – 352 с.

80. Кузнецов С.Д. Введение в информационные системы / С.Д. Кузнецов // СУБД. - 1997. / №2. / С.83-96.

81. Кузнецов С.Д. Введение в СУБД: Ч.3 / С.Д. Кузнецов // Системы управления базами данных. – 1995. №3. - С.114-127.

82. Кузнецов С.Д. Введение в СУБД: Ч.4 / Кузнецов С.Д. // Системы управления базами данных. – 1995. №4. - С.114-122.

83. Кузнецов С.Д. Введение в СУБД: Ч.5 / С.Д. Кузнецов // Системы управления базами данных. – 1996. №1. - С.124-143.

84. Кузнецов С.Д. Введение в СУБД: Ч.6 / С.Д. Кузнецов // Системы управления базами данных. – 1996. №2. - С.130-140.

85. Кузнецов С.Д. Введение в СУБД: Ч.8 / С.Д. Кузнецов // Системы управления базами данных. – 1996. №4. - С.98-119.

86. Кузнецов С.Д. Введение в СУБД: Ч.9 / С.Д. Кузнецов // Системы управления базами данных. – 1996. №5-6. - С.136-153.

87. Кузнецов С.Д.. Введение в системы управления базами данных / С.Д. Кузнецов // СУБД. 1995. №1. С.117-127.

88. Кузнецов С.Д.. Введение в СУБД: Ч.2 / С.Д. Кузнецов // СУБД. – 1995. - №2. - С.116-124.

89. Куправа Т.А. Создание и программирование баз данных средствами СУБД./ Т.А. Куправа – М.: Мир, 2003.

90. Леоненков, А. В. Объектно-ориентированный анализ и проектирование с использованием UML и IBM Rational Rose: учеб. пособие / А. В. Леоненков. - М. : Интернет-Ун-т Информ. Технологий: БИНОМ. Лаборатория знаний, 2006. - 320 с. - (Основы информационных технологий). - Библиогр.: с. 317-318.

91. Маклаков С.В. Bpwin и Erwin. CASE – средство разработки информационных систем. / С.В. Маклаков – М.: ДИАЛОГ – МИФИ, 2000 –256 с.

92. Маклаков С.В. Инструментальные средства создания корпоративных информационных систем / С.В. Маклаков // Компьютер Пресс, №7-№9, 2004.

93. Манифест объектно-ориентированных баз данных / Аткинсон М., Бансилон Ф. [и др.] // СУБД. – 1995. - №4. –С. 142-155.

94. Мартин Дж. Введение в SQL / Мартин Дж., Грабер. - М., изд Лори, 1996.

95. Мартин Дж. Организация баз данных в вычислительных системах. / Дж. Мартин. -М.: Мир, 1980.

96. Махмутова М.В. Введение в технологии баз данных. Учебное пособие. / М.В.Махмутова, Э.Р.Ипатова. - Магнитогорск: МаГУ, 2007.-301 с. (Рекомендовано УМО по прикладной информатике)

97. Метатехнология IDEF1X. Стандарт. Русская версия. – М.: Метатехнология, 1993.

98. Нагао М. Структуры и базы данных: Пер. с япон. / М. Нагао, Т. Катаяма, С. Уэмура – М.: Мир,2006. – 197 с.

99. Назарова О.Б. Практикум по дисциплине «Теория экономических информационных систем» : учеб. пособие./ Э.Р. Ипатова, О.Б.Назарова, О.Е. Масленникова. - Магнитогорск: МаГУ, 2007. - 114 с. (Рекомендовано УМО по прикладной информатике)

100. Наумов А.Н. Системы управления базами данных и знаний: Справ. изд. / А.Н. Наумов, А.М. Вендров, В.К. Иванов и др. - М.: Финансы и статистика, 2001.

101. Ньюмен У.М. Основы интерактивной машинной графики. / У.М. Ньюмен, Р.Ф. Спруэлл - М.: Мир, 1976.

102. Олле Т.В. Предложения КОДАСИЛ по управлению базами данных / Т.В. Олле. Пер. с англ. В.И.Филиппова и С.М. Круговой. – М.: Финансы и статистика, 1981. – 286 с.

103. Острейковский В.А. Информатика: Учеб для ВУЗов. - М.:Высш. шк., 2008.-511с.-Рекомендовано Министерством образования и науки.

104. Роб, П. Системы баз данных: проектирование, реализация и управление. – 5-е изд., перераб. И доп.: Пер. с англ. / П. Роб, К. Коронел. – СПб.: БХВ-Петербург, 2004. – 1040с.: ил.

105. Симонович С.В., Евсеев Г.А., Алексеев А.Г. Специальная информатика: Учебное пособие. - М.: АСТ-ПРЕСС: Инфорком-Пресс, 2008.-480

106. Системы управления базами данных для ЕС ЭВМ: Справочник / Под общ. ред. В.М.Савинкова. М.: Финансы и статистика, 1984.

107. Советский энциклопедический словарь / Под ред. А.М. Прохорова. - М.: Советская энциклопедия, 1989.

108. Солтон Дж. Динамические библиотечно-информационные системы. / Дж. Солтон - М.: Мир, 1979.

109. Станек Уильман Р. Microsoft SQL Server 2005. Справочник администратора/ Пер. с англ.- М.: Русская редакция, 2008.- 544 с.: ил.*

110. Тиори Т. Проектирование структур баз данных: В 2кн.: Пер с англ./ Т. Тиори, Дж. Фрай – М.: Мир, 1985.

111. Тощев А.Н. ERwin и автоматическая генерация кода клиентских приложений / А.Н. Тощев // Компьютер Пресс, №10, 1998.

112. Фролов А. Базы данных в Интернете. Практич. рук – во по созданию Web – приложений с базами данных. – М., 2000.

113. Хаббард Дж. Автоматизированное проектирование баз данных./ Дж. Хаббард – М.: Мир, 2004. – 292 с.

114. Хеллман О. Введение в теорию оптимального поиска./ О. Хеллман - М.: Наука, 2005.

115. Хомоненко А. Д. Базы данных: Учебник для высших учебных заведений / А.Д. Хомоненко, В.М. Цыганков, М.Г. Мальцев – СПб.:КОРОНА принт, 2002. – 672 с.

116. Чамберлин Д.Д. и др. SEQUEL 2: унифицированный подход к определению, манипулированию и контролю данных // Д.Д. Чемберлин [и др.]. СУБД.-1996. - №1. – С. 144-159.

117. Четвериков В.Н. Базы и банки данных. / В.Н. Четвериков, Г.И. Ревунков, Т.М. Самохвалов – М.: Высшая школа, 2003. – 248 с.

ПРИЛОЖЕНИЯ

Приложение 1

Ключевые слова, определенные в стандарте ANSI SQL, которые не могут быть использованы в качестве имен объектов баз данных (таблиц, полей, имен пользователей)

ABSOLUTE	CROSS	GET	NEXT	SPACE
ACTION	CURRENT	GLOBAL	NO	SQL
ADD	CURRENT_DATE	GO	NOT	SQLCODE
ALL	CURRENT_TIME	GOTO	NULL	SQLERROR
ALLOCATE	CURRENT_TIMESTAMP	GRANT	OCTET_LENGTH	SQLSTATE
ALTER	CURRENT_USER	GROUP	OF	SUBSTRING
AND	CURSOR	HAVING	ON	SUM
ANY	DATE	HOUR	ONLY	SYSTEM_USER
ARE	DAY	IDENTITY	OPEN	TABLE
AS	DEALLOCATE	IMMEDIATE	OPTION	TEMPORARY
ASC	DEC	IN	OR	THEN
ASSERTION	DECIMAL	INDICATOR	ORDER	TIME
AT	DECLARE	INITIALLY	OUTER	TIMESTAMP
AUTHORIZATION	DEFAULT	INNER	OUTPUT	TIMEZONE_HOUR
AVG	DEFERRABLE	INPUT	OVERLAPS	TIMEZONE_MINUTE
BEGIN	DEFERRED	INSENSITIVE	PAD	TO
BETWEEN	DELETE	INSERT	PARTIAL	TRAILING
BIT	DESC	INT	POSITION	TRANSACTION
BIT_LENGTH	DESCRIBE	INTEGER	PRECISION	TRANSLATE
BOTH	DESCRIPTOR	INTERSECT	PREPARE	TRANSLATION
BY	DIAGNOSTICS	INTERVAL	PRESERVE	TRIM
CASCADE	DISCONNECT	INTO	PRIMARY	TRUE
CASCADED	DISTINCT	IS	PRIOR	UNION
CASE	DOMAIN	ISOLATION	PRIVILEGES	UNIQUE
CAST	DOUBLE	JOIN	PROCEDURE	UNKNOWN
CATALOG	DROP	KEY	PUBLIC	UPDATE
CHAR	ELSE	LANGUAGE	READ	UPPER

CHARACTER	END	LAST	REAL	USAGE
CHAR_LENGTH	END-EXEC	LEADING	REFERENCES	USER
CHARACTER_LENGTH	ESCAPE	LEFT	RELATIVE	USING
CHECK	EXCEPT	LEVEL	RESTRICT	VALUE
CLOSE	EXCEPTION	LIKE	REVOKE	VALUES
COALESCE	EXEC	LOCAL	RIGHT	VARCHAR
COLLATE	EXECUTE	LOWER	ROLLBACK	VARYING
COLLATION	EXISTS	MATCH	ROWS	VIEW
COLUMN	EXTERNAL	MAX	SCHEMA	WHEN
COMMIT	EXTRACT	MIN	SCROLL	WHENEVER
CONNECT	FALSE	MINUTE	SECOND	WHERE
CONNECTION	FETCH	MODULE	SECTION	WITH
CONSTRAINT	FIRST	MONTH	SELECT	WORK
CONSTRAINTS	FLOAT	NAMES	SESSION	WRITE
CONTINUE	FOR	NATIONAL	SESSION_USER	YEAR
CONVERT	FOREIGN	NATURAL	SET	ZONE
CORRESPONDING	FOUND	NCHAR	SIZE	
COUNT	FROM	NULLIF	SMALLINT	
CREATE	FULL	NUMERIC	SOME	

Приложение 2

Список потенциальных ключевых слов, зарезервированных для последующих версий стандарта SQL

AFTER	EQUALS	OLD	RETURN	TEST
ALIAS	GENERAL	OPERATION	RETURNS	THERE
ASYNC	IF	OPERATORS	ROLE	TRIGGER
BEFORE	IGNORE	OTHERS	ROUTINE	TYPE
BOOLEAN	LEAVE	PARAMETERS	ROW	UNDER
BREADTH	LESS	PENDANT	SAVEPOINT	VARIABLE
COMPLETION	LIMIT	PREORDER	SEARCH	VIRTUAL
CALL	LOOP	PRIVATE	SENSITIVE	VISIBLE
CYCLE	MODIFY	PROTECTED	SEQUENCE	WAIT
DATA	NEW	RECURSIVE	SIGNAL	WHILE
DEPTH	NONE	REF	SIMILAR	WITHOUT
DICTIONARY	OBJECT	REFERENCING	SQLEXCEPTION	
EACH	OFF	REPLACE	SQLWARNING	
ELSEIF	OID	RESIGNAL	STRUCTURE	

Приложение 3

Математические и строковые функции

Функция	Назначение
ABS	Возвращает абсолютное значение числа
CEIL	Округляет дробное число
FLOOR	Удаляет дробную часть числа
GREATEST	Возвращает наибольшее из двух значений. Эта функция используется в Microsoft Access и Microsoft SQL Server
LEAST	Возвращает наименьшее из двух значений. Эта функция используется в Microsoft Access и Microsoft SQL Server
MOD	Возвращает остаток от деления одного числа на другое
POWER	Возвращает значение, равное одному числу в степени, равной другому числу
ROUND	Округляет число с точностью до указанного десятичного знака
SIGN	Возвращает -1, если число отрицательное, и 1, если положительное
SQRT	Вычисляет квадратный корень числа
LEFT	Возвращает указанное число знаков строки, начиная слева. Эта функция используется в Microsoft Access и Microsoft SQL Server
RIGHT	Возвращает указанное число знаков строки, начиная справа. Эта функция используется в Microsoft Access и Microsoft SQL Server
UPPER	Заменяет все буквы в строке на прописные
LOWER	Заменяет все буквы в строке на строчные
INITCAP	Расставляет заглавные буквы в начале слов в строке
LENGTH	Вычисляет число символов в строке
LPAD	Добавляет указанный символ в левую часть строки в количестве, необходимом для того, чтобы строка имела заданную длину
RPAD	Добавляет указанный символ в правую часть строки в количестве, необходимом для того, чтобы строка имела заданную длину
SUBSTR	Извлекает подстроку нужной длины из строки, начиная с указанной позиции

ALTER DATABASE	ALTER PROCEDURE	ALTER TABLE
ALTER TRIGGER	ALTER VIEW	CREATE DATABASE
CREATE DEFAULT	CREATE INDEX	CREATE PROCEDURE
CREATE RULE	CREATE SCHEMA	CREATE TABLE
CREATE TRIGGER	CREATE VIEW	DENY
DISK INIT	DISK RESIZE	DROP DATABASE
DROP DEFAULT	DROP INDEX	DROP PROCEDURE
DROP RULE	DROP TABLE	DROP TRIGGER
DROP VIEW	GRANT	LOAD DATABASE
LOAD LOG	RESTORE DATABASE	RESTORE LOG
REVOKE	RECONFIGURE	
TRUNCATE TABLE	UPDATE STATISTICS	

Операторы языка SQL

1. Операторы описания

```
CREATE DATABASE database-name                создание базы данных
[WITH
{ [BUFFERED] LOG                  с (буферизованной) журнализацией
| LOG MODE ANSI}]                       в стандарте ANSI
CREATE SCHEMA schema-name             создание схемы базы данных
CREATE [TEMP] TABLE table-name        создание таблицы
{ с именем table-name
{ column-name column-type             имя и тип столбца
| column-name {BYTE | TEXT}                 типа BYTE | TEXT
[IN {TABLE | blobspace-name}]         где создавать
[NOT NULL]                            отсутствие NULL-значений
[UNIQUE [(unique col-list)]           уникальность
[CONSTRAINT constraint-name]          наложено ограничение
[, ...])
[WITH NO LOG]                              без журнализации
[IN dbspace-name]                     где создавать
[LOCK MODE ({PAGE | ROW})]            уровень блокирования
CREATE [UNIQUE][CLUSTER]              создание индекса
INDEX index-name ON table-name        для какой таблицы
(column-name [ASC | DESC] [,...])     по какому столбцу
                                      и в каком порядке

CREATE SYNONYM                             создание синонима имени
synonym-name FOR table-name           указанной таблицы
CREATE VIEW view-name                 создание представления
[(column-list)] AS SELECT-statement
ALTER TABLE table-name                изменение структуры таблицы
{
ADD ( newcol-name                     если необходимо добавить
newcol-type [NOT NULL][UNIQUE         столбец в таблицу
[CONSTRAINT constraint-name]]         наложено ограничение
[, ...])
[BEFORE oldcol-name]                  перед каким столбцом вставлять
| DROP( oldcol-name [, ...] )         удалить столбец(цы)
| MODIFY (oldcol-name newcol-type
NOT NULL] [, ...]
| ADD CONSTRAINT UNIQUE ( oldcol-name [, ...] )
[CONSTRAINT constraint-name]
| DROP CONSTRAINT (constraint-name [, ...] )
}
CLOSE DATABASE                  закрытие текущей базы данных
```

DATABASE database-name активизация базы данных
[EXCLUSIVE] [в монопольном режиме]
CONNECT TO database-name активизация базы данных
USER user-name USING parol имя пользователя и пароль
DROP DATABASE удаление базы данных
{database-name | char-variable } по ее имени либо по переменной

DROP INDEX index-name удаление индекса
DROP TABLE table-name удаление таблицы из базы данных
DROP SYNONYM удаление синонима
DROP VIEW view-name удаление представления
RENAME TABLE oldname TO newname переименование таблицы
RENAME COLUMN переименование столбца
oldcol-name TO newcol-name

2. Операторы манипулирования данными

DELETE FROM table-name удаление строк из таблицы
[WHERE }]
{condition по указанному условию
INSERT вставка строк в таблицу
INTO table-name [(column-list)] имена таблиц и столбцов
{VALUES(value-list) список значений полей
| SELECT-statement} вставка результата выполнения оператора SELECT

SELECT выборка данных из таблиц
[ALL | [DISTINCT | UNIQUE]] select-list что выбирается
[INTO variable-list] куда выбирается (ESQL/C)
FROM откуда выбирается
{ table-name [table-alias] из указанных таблиц
| OUTER table-name [table-alias] для создания
}[,....] внешнего соединения
[WHERE condition] условие выбора строк
[GROUP BY column-list] группирование
[HAVING condition] условие в группах
[ORDER BY column-list [ASC | DESC][,...]] сортировка
[INTO TEMP table-name] куда поместить результат
UPDATE table-name SET модификация строки таблицы
{column-name=expr[, ...] } имена столбцов
[WHERE {condition условие изменения
|CURRENT OF cursor-name}] имя курсора
LOAD FROM "pathname" загрузка базы данных из файла
[DELIMITER "char"] разделитель полей
{INSERT INTO
table-name[(column-name [, ...])] | insert-statement } имя таблицы и столбцы

UNLOAD выгрузка базы данных в файла
SELECT в ASCII-файл
TO "pathname" [DELIMITER "char"] путь к файлу и разделитель полей
SELECT-statement оператор SELECT

3. Операторы определения транзакций

BEGIN WORK определение начала транзакции
COMMIT WORK подтверждение транзакции
ROLLBACK WORK откат транзакции

4. Операторы определения прав доступа

GRANT установка уровня привилегий к таблице
{ ALL все привилегии
| INSERT привилегии на вставку
| DELETE привилегии на удаление
| SELECT привилегии на просмотр
| UPDATE привилегии на изменение строк
| REFERENCES привилегии на установку ограничений по ссылке на столбцы
| INDEX привилегии на построение индекса
| ALTER привилегии на изменение структуры таблицы
}
ON table-name имя таблицы
TO {PUBLIC | user-list} кому передаются права
[WITH GRANT OPTION] с правом передачи этих прав
GRANT установка уровня привилегий к базе данных
{CONNECT привилегии на запросы и обновление данных
| RESOURCE привилегии на изменение структуры базы данных
| DBA привилегии администратора, за исключением возможности менять системную таблицу systables
}
TO {PUBLIC | user-list} кому передаются права
REVOKE отмена привилегий на пользование таблицей или базой данных
{table-privilege ON table-name имя таблицы
|database-privilege } привилегия на базу данных
FROM {PUBLIC | user-list} список пользователей
LOCK TABLE table-name блокировка таблицы
IN { SHARE в разделяемом режиме
| EXCLUSIVE} в монопольном режиме
MODE
UNLOCK TABLE table-name снятие блокировки с таблицы
SET ISOLATION TO установка уровня изоляции

{ CURSOR STABILITY — по стабильному курсору
| DIRTY READ — грязное чтение
| COMMITTED READ — подтвержденное чтение
| REPEATABLE READ } — повторяемое чтение
SET LOCK MODE TO — установка режима доступа
{ NOT WAIT | WAIT [seconds]} — ждать/ не ждать освобождения блокированного ресурса

5. Встроенный SQL

DECLARE cursor-name [SCROLL] CURSOR — определение курсора
FOR SELECT-statement — ассоциированный оператор Select
| operator-name — или динамически подготовленный оператор
[INTO host-name] — куда выбирать
OPEN cursor-name — открытие курсора
[USING host-name] — главные переменные
FETCH [parameter-list] cursor-name — выбор данных по курсору
[INTO host-name] — куда выбирать
CLOSE cursor-name — закрытие курсора
PREPARE operator-name FROM — подготовка динамического
char-string — оператора из символьной строки
EXECUTE operator-name — выполнение динамического SQL
[USING host-name] — главные переменные

6. Триггеры и процедуры

CREATE TRIGGER trigger-name — создание триггера
{INSERT |DELETE | UPDATE | — условие включения триггера
UPDATE OF column-name} ON table-name — имя таблицы
[{REFERENCING NEW AS correlation-name} | — имя переменной с удаляемой,
{REFERENCING OLD AS correlation -name} | — вставляемой, модифицируемой
{REFERENCING NEW AS correlation -name — строкой (только с For each row)
OLD AS correlation -name }
{BEFORE | FOR EACH ROW |AFTER} — момент применения триггера
[WHEN (condition)] — дополнительное условие
{INSERT-statement | DELETE-statement | — SQL-оператор или хранимая
UPDATE-statement | — процедура, выполняемые
EXECUTE PROCEDURE procedure-name} — триггером
DROP TRIGGER trigger-name — удаление триггера из базы данных

CREATE PROCEDURE procedure-name — создание процедуры
([expression[,...]]) — список аргументов процедуры
RETURNING type — тип возвращаемого значения
[define-stmt-list] — локальные переменные
[exception-declaration] — конструкции условий

[statement-list]	выполняемые операторы
END PROCEDURE	процедуры
CALL procedure-name	вызов одной процедуры из другой
([expression[,...]])	параметры процедуры
[RETURNING var_name]	возвращаемое значение
EXECUTE PROCEDURE procedure-name	вызов процедуры из клиентской программы или триггера
([expression[,...]])	параметры процедуры
DROP PROCEDURE procedure-name	удаление процедуры из базы данных

Учебное текстовое электронное издание

Махмутова Марина Владимировна

ВВЕДЕНИЕ В ТЕХНОЛОГИИ БАЗ ДАННЫХ

Учебное пособие

1,81 Мб
1 электрон. опт. диск

г. Магнитогорск, 2015 год
ФГБОУ ВПО «МГТУ»
Адрес: 455000, Россия, Челябинская область, г. Магнитогорск,
пр. Ленина 38

ФГБОУ ВПО «Магнитогорский государственный
технический университет им. Г.И. Носова»
Кафедра прикладной информатики
Центр электронных образовательных ресурсов и
дистанционных образовательных технологий
e-mail: ceor_dot@mail.ru