Э. В. Фуфаев
Д. Э. Фуфаев

РАЗРАБОТКА И ЭКСПЛУАТАЦИЯ УДАЛЕННЫХ БАЗ ДАННЫХ

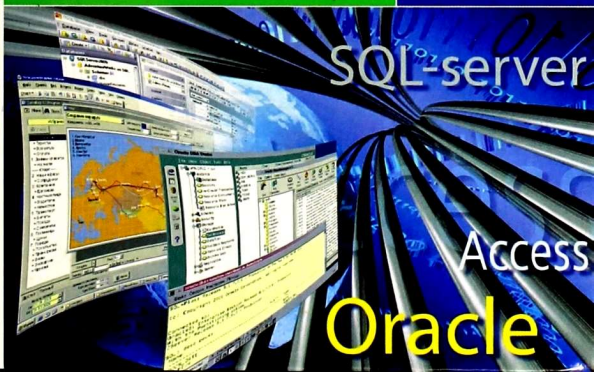
4-е издание

СРЕДНЕЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАНИЕ

УЧЕБНИК

ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ACADEMA



СРЕДНЕЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАНИЕ

Э. В. ФУФАЕВ, Д. Э. ФУФАЕВ

РАЗРАБОТКА И ЭКСПЛУАТАЦИЯ УДАЛЕННЫХ БАЗ ДАННЫХ

УЧЕБНИК

Рекомендовано
Федеральным государственным учреждением
«Федеральный институт развития образования»
в качестве учебника для студентов среднего
профессионального образования по специальности
230105 «Программное обеспечение вычислительной техники
и автоматизированных систем»

Регистрационный номер рецензии 583 от 28 июля 2009 г. ФГУ «ФИРО»

4-е издание, стереотипное

ACADEMA
Москва
Издательский центр «Академия»
2014

УДК 621.391(075.32)
ББК 32.81я723
Ф964

Рецензенты:
преподаватель Московского государственного колледжа информационных технологий *И. А. Кумскова*;
зам. декана факультета «Информатика и телекоммуникации» Московского государственного института электроники и математики, доц., канд. техн. наук *Д. П. Николаев*

Фуфаев Э. В.

Ф964 Разработка и эксплуатация удаленных баз данных : учебник для студ. учреждений сред. проф. образования / Э. В. Фуфаев, Д. Э. Фуфаев. — 4-е изд., стер. — М. : Издательский центр «Академия», 2014. — 256 с.

ISBN 978-5-4468-0467-2

Даны теоретические основы и практические рекомендации по разработке и эксплуатации удаленных баз данных при создании информационных систем для различных задач управления. Рассмотрены современные технологии доступа к удаленным базам данных.

Для студентов учреждений среднего профессионального образования. Может быть полезен для специалистов, работающих в области управления производством и бизнесом.

УДК 621.391(075.32)
ББК 32.81я723

ISBN 978-5-4468-0467-2

ПРЕДИСЛОВИЕ

В настоящее время целью экономической деятельности любой фирмы является получение прибыли. Эффективность экономической деятельности зависит от того, как и за счет чего будет получена прибыль, т.е. от эффективности удовлетворения потребностей заказчиков товаров и услуг. Решение этой задачи несомненно связано с обязательным выполнением требований международных стандартов ISO 1900:2000.

Достижение поставленной цели — задача многоплановая, она зависит от творческих способностей всех сотрудников фирмы. Решить эту задачу можно лишь в том случае, если каждый участник производственного процесса от руководителя до рядового исполнителя сможет принимать оптимальные решения на своем участке производства.

Принимать оптимальные решения с первого раза, а не исправлять последствия неправильных решений, — вот главная задача любого специалиста, для эффективного решения которой можно использовать компьютерные информационные системы, разрабатываемые на основе баз данных (БД).

Базы данных как одно из направлений теории информации представляют собой методы и средства разработки компьютерных информационных систем, основу которых составляют особым образом структурированные файлы, предоставляющие пользователю эффективные методы получения и анализа данных, необходимых для принятия оптимального решения.

Теория разработки баз данных — сравнительно молодая область науки, однако на сегодняшний день базы данных являются основой таких направлений в разработке автоматизированных систем обработки информации, как системы искусственного интеллекта, экспертные системы, системы автоматизированного конструкторского и технологического проектирования.

Учебник состоит из шести частей.

Часть I посвящена теоретическим основам проектирования удаленных баз данных.

В части II изложены:

методы разработки и управления удаленными базами данных с применением языка SQL, системы SQL Server2000 и СУБД Oracle;

технологии доступа к удаленным базам данных: ODBC, COM, ADO.NET, .NET FrameWork, CORBA, MIDAS.

В части III изложены методы проектирования серверной части приложения баз данных:

концептуальное, логическое и физическое проектирование базы данных;

технологии проектирования и модификации таблиц командами языка SQL;

преобразование проекта базы данных формата Microsoft Access в формат SQL Server;

разработку пользовательских представлений, хранимых процедур и триггеров.

В части IV рассмотрены методы и средства проектирования клиентской части приложения баз данных:

автоматизация работы с данными визуальными средствами MS Access;

разработка программ управления удаленными базами данных с применением внедренных операторов SQL;

применение Web-технологий в разработке удаленных баз данных.

В части V рассмотрены методы администрирования и эксплуатации удаленных баз данных:

защиту информации;

восстановление данных;

управление буферами базы данных;

резервное копирование базы данных.

В части VI рассмотрены некоторые направления развития постреляционных систем управления удаленными базами данных. Дан пример системы управления жизненным циклом продукции.

Данный учебник отражает многолетний опыт работы авторов в этой области информационных технологий, в том числе 10-летний опыт преподавания в Московском авиационном технологическом институте — Российском государственном технологическом университете им. К. Э. Циолковского.

ВВЕДЕНИЕ

Когда мы только начинали создавать информационные системы с применением первых версий системы Microsoft Access, нас удивлял перевод назначения этой системы управления базами данных (СУБД) как настольной: стол с возможностью хранения миллионов записей! Одна из первых наших коммерческих разработок в конце 1990-х годов была направлена на создание в среде Microsoft Access 2.0 информационной системы для частной фирмы, занимающейся поставкой комплектующих изделий обувным фабрикам России. Причем разработанная система предназначалась для совместного использования различными специалистами, для чего применялись процедуры импорта, экспорта и присоединения данных на основе стандарта ODBC (Open DataBase Connectivity — открытый доступ к данным). Фактически мы продублировали отдельные компоненты данной системы на «рабочие столы» специалистов, объединенных между собой локальной сетью, организовав при этом как бы удаленные базы данных.

Под удаленными следует понимать такие базы данных, доступ к информации и управление которыми осуществляются с помощью линий связи.

В разработанной нами системе с помощью линий связи осуществлялся только *доступ к информации*, поэтому мы и назвали их *как бы удаленные базы данных*.

На сегодняшний день существуют следующие понятия баз данных: настольные, многопользовательские, удаленные и распределенные, которые можно подразделить на две группы: *однопользовательские* (настольные) и *многопользовательские* (удаленные).

Таким образом, наше удивление по поводу термина *настольная* было преждевременным. Казалось также, что можно назвать данную систему локальной, однако в то время уже существовало понятие ЛВС — локальной вычислительной сети, объединяющей в единое физическое пространство отдельные компьютеры пользователей, а первые версии Microsoft Access непосредственно не предназначались для организации многопользовательских баз данных.

Из всего сказанного можно сделать следующие выводы.

1. Понятия *многопользовательская база данных* и *удаленная база данных* можно считать тождественными.

Сущность CALS-технологий сводится к созданию *единого информационного пространства (ЕИП), обеспечивающего возможность получения достоверной информации, необходимой для принятия оптимальных решений в течение всего жизненного цикла изделия, начиная с его проектирования и заканчивая утилизацией после окончания сроков эксплуатации.*

За время своего существования расшифровка аббревиатуры CALS претерпела ряд изменений.

В момент возникновения аббревиатура CALS расшифровывалась как Computer Aided Logistic Support — компьютерная поддержка поставок и логистики, т.е. в то время акцент делался на применение компьютеров для управления процессами поставок, транспортировки и эксплуатации продукции.

С течением времени, когда применение компьютеров стало обычным делом, понятие CALS трансформировалось в Continuous Acquisition and Life cycle Support — непрерывная информационная поддержка поставок и жизненного цикла продукции. Здесь акцент смещен, во-первых, в сторону непрерывности взаимодействия заказчика и поставщика в ходе процессов поставки продукции, а во-вторых, в сторону охвата всего ее ЖЦ.

По умолчанию предполагается, что речь идет о сложной наукоемкой продукции, которая требует создания, преобразования и передачи между различными участниками ЖЦ больших объемов технической информации.

В последнее время появилась еще одна трактовка понятия CALS: Commerce At Light Speed — высокоскоростная (быстрая) коммерция.

Эта трактовка связана с постоянно расширяющейся сферой применения электронного бизнеса (e-business) и электронной коммерции (e-commerce), суть которых состоит в том, что коммерческие сделки заключаются посредством глобальной сети Интернет. В ходе этих сделок стороны обмениваются данными (нередко значительного объема) в электронном безбумажном виде, при необходимости скрепляя эти данные электронными цифровыми подписями (ЭЦП), имеющими такой же юридический статус, как и собственноручная подпись.

Электронный обмен данными происходит в темпе, не возможном при традиционных способах общения.

Рассмотрим в качестве примеров организации работ с удаленными базами данных небольшую фирму «Бюро переводов» и коммерческий банк.

На рис. В.1 показана схема организации локальной вычислительной сети *Бюро переводов*, из которой видно, что работа с удаленными клиентами, в качестве которых выступают как ее сотрудники (переводчики), так и заказчики работ, осуществляется через сеть Интернет.

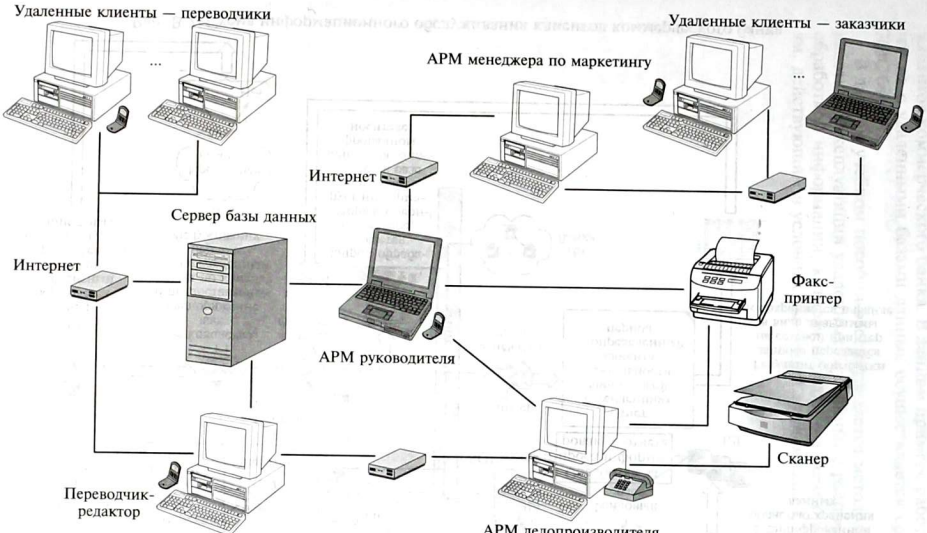


Рис. В.1. Схема организации локальной вычислительной сети *Бюро переводов* (АРМ — автоматизированное рабочее место)

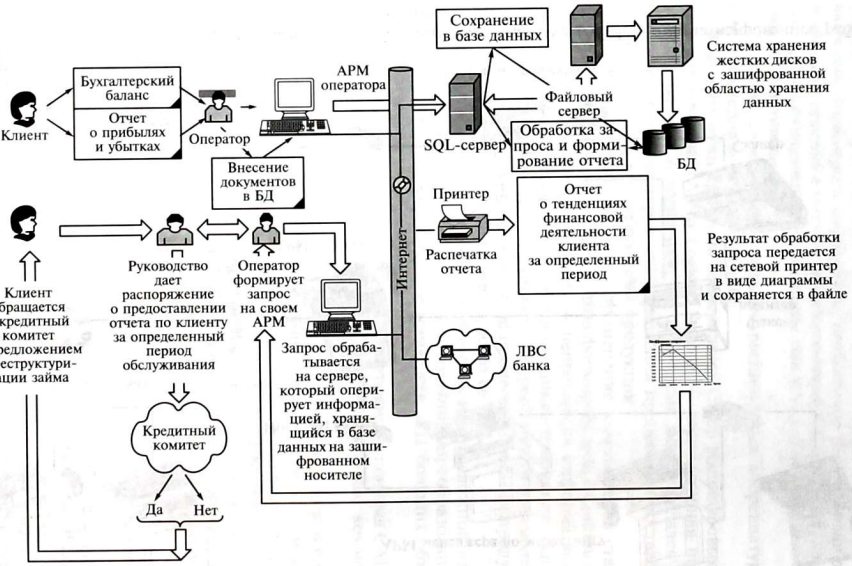


Рис. В.2. Схема информационного обслуживания клиентов коммерческого банка

На рис. В.2 показана схема информационного обслуживания клиентов коммерческого банка. В данном примере работа сотрудников с удаленными базами данных осуществляется средствами ЛВС банка.

В данном учебном пособии рассматривается методология разработки и эксплуатации удаленных баз данных с распределенной обработкой информации, которая необходима всем организациям, действующим в условиях рыночной экономики.

ЧАСТЬ I

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ПРОЕКТИРОВАНИЯ УДАЛЕННЫХ БАЗ ДАННЫХ

ГЛАВА 1

АРХИТЕКТУРЫ УДАЛЕННЫХ БАЗ ДАННЫХ

1.1. Термины и определения

Системы управления удаленными (распределенными) базами данных — это СУБД (СУРБД), обеспечивающие возможность одновременного доступа к информации различным пользователям.

Рассмотрим термины, применяемые в системах управления распределенными базами данных.

Архитектура БД — организация взаимодействия аппаратных средств.

Виды архитектуры БД: клиент—сервер, двухуровневая и трехуровневая клиент-сервер, файл—сервер.

Архитектура ODBC (Open DataBase Connectivity) — открытый интерфейс доступа к базам данных, т.е. взаимодействие процессора (ядра) базы данных Jet с внешними источниками данных.

Модели данных — схемы, характеризующие базы данных с разных сторон с целью определить оптимальное построение информационной системы.

Ядро базы данных — внутренняя структура СУБД, обеспечивающая доступ ко всем компонентам базы данных. В новых версиях СУБД Access называется Microsoft Data Engine (MSDE); в ранних версиях ядро базы данных называлось *машина базы данных Microsoft Jet*. Ядро базы данных обеспечивает поддержку символов различных алфавитов, синтаксис языка SQL и другие средства обработки различных типов данных.

Пользователь БД — программа или человек, обращающийся к базе данных.

Запрос — процесс обращения пользователя к БД с целью ввести, получить или изменить информацию.

Транзакция — последовательность операций модификации данных в БД, переводящая ее из одного непротиворечивого состояния в другое непротиворечивое состояние.

Логическая структура БД — определение БД на физически независимом уровне, что ближе всего соответствует концептуальной ее модели.

Топология БД, или *структура распределенной БД*, — схема распределения физической организации базы данных в сети.

Локальная автономность — понятие, означающее, что информация локальной БД и связанные с ней определения данных принадлежат локальному владельцу и им управляются.

Удаленный запрос — запрос к базам данных, находящихся на ресурсах локальной сети предприятия или сети Интернет.

Возможность реализации удаленной транзакции — обработка одной транзакции, состоящей из множества SQL-запросов, на одном удаленном узле.

Поддержка распределенной транзакции — обработка транзакции, состоящей из нескольких SQL-запросов, выполняемых на нескольких узлах сети (удаленных или локальных), но каждый из которых обрабатывается только на одном узле.

Распределенный запрос — запрос, при обработке которого используются данные из БД, расположенные в разных узлах сети.

Системы распределенной обработки данных в основном связаны с первым поколением БД, которые строились на мультипрограммных операционных системах, хранились на устройствах внешней памяти центральной ЭВМ и использовали терминальный многопользовательский режим доступа. При этом пользовательские терминалы не имели собственных ресурсов, т.е. процессоров и памяти, которые могли бы использоваться для хранения и обработки данных. Первой полностью реляционной системой, работающей в многопользовательском режиме, была СУБД SYSTEM R фирмы IBM. Именно в ней были реализованы как язык манипулирования данными SQL, так и основные принципы синхронизации, применяемые при распределенной обработке данных, которые до сих пор являются базисными практически во всех коммерческих СУБД.

1.2. Архитектуры клиент—сервер в технологии управления удаленными базами данных

Вычислительная модель клиент-сервер исходно связана с появлением открытых систем в 1990-х гг. Термин *клиент—сервер* применялся к архитектуре программного обеспечения, состоящего из двух процессов обработки информации: клиентского и серверного. Клиентский процесс запрашивал некоторые услуги, а серверный — обеспечивал их выполнение. При этом предполагалось, что один серверный процесс может обслужить множество клиентских процессов. Учитывая, что аппаратная реализация этой моде-

ли управления базами данных связана с созданием локальных вычислительных сетей предприятия, такую организацию процесса обработки информации называют архитектурой клиент—сервер.

Основной принцип модели клиент—сервер применительно к технологии управления базами данных заключается в разделении функций стандартного интерактивного приложения на пять групп, имеющих различную природу:

- функции ввода и отображения данных (Presentation Logic);
- прикладные функции, определяющие основные алгоритмы решения задач приложения (Business Logic);
- функции обработки данных внутри приложения (DataBase Logic);
- функции управления информационными ресурсами (DataBase Manager System);
- служебные функции, играющие роль связок между функциями первых четырех групп.

Структура типового приложения, работающего с базой данных в архитектуре клиент—сервер, приведена рис. 1.1.

Как видно из рис. 1.1, клиентская часть приложения включает в себя следующие части:

- презентационную логику;
- бизнес-логику, или логику собственно приложений;
- логику обработки данных;
- процессор управления данными.

Презентационная логика (Presentation Logic) как часть приложения определяется тем, что пользователь видит на своем экране, что приложение работает. Сюда относятся все интерфейсные экранные формы, которые пользователь видит или заполняет в ходе работы приложения, а также все то, что выводится пользователю на экран в качестве результатов решения некоторых про-

Рис. 1.1. Структура типового приложения, работающего с базой данных

межуточных задач либо как справочная информация. Следовательно, основными задачами презентационной логики являются:

- формирование экранных изображений;
- чтение и запись в экранные формы информации;
- управление экраном;
- обработка движений мыши и нажатие клавиш клавиатуры.

Бизнес-логика, или логика собственно приложений (Business Processing Logic), — это часть кода приложения, которая определяет собственно алгоритмы решения конкретных его задач. Обычно этот код записывается с использованием различных языков программирования, таких как C, C++, Visual Basic и др.

Логика обработки данных (Data Manipulation Logic) — это часть кода приложения, которая непосредственно связана с обработкой данных внутри него. Данными управляет собственно СУБД, а для обеспечения доступа к ним используется язык SQL.

Процессор управления данными (DataBase Manager System Processing) — это собственно СУБД, которая обеспечивает хранение и управление базами данных. В идеале функции СУБД должны быть скрыты от бизнес-логики приложения, однако при рассмотрении архитектуры приложения мы выделим их в отдельную его часть.

Модели распределений	Компоненты приложения						Пользователь
	Функции логики представления DP		Функции бизнес-логики		Функции управления данными		
			RP	RBL	RDM	DDM	
Распределенное представление (DP)	■	■					Клиент
		■	■	■	■	■	Сервер
Удаленное представление (RP)	■						Клиент
		■	■	■	■	■	Сервер
Распределенная бизнес-логика (RBL)	■	■	■				Клиент
				■	■	■	Сервер
Удаленное управление данными (RDM)	■	■	■	■			Клиент
					■	■	Сервер
Распределенное управление данным (DDM)	■	■	■	■	■		Клиент
						■	Сервер
Совмещение RBL и DDM	■	■	■		■		Клиент
				■		■	Сервер

Рис. 1.2. Распределение функций компонентов приложения в моделях клиент—север

В централизованной архитектуре (Host-Based Processing) указанные части приложения располагаются в единой среде и комбинируются внутри одной исполняемой программы. В децентрализованной архитектуре эти части приложения могут быть по-разному распределены между серверным и клиентским процессами.

В зависимости от характера распределений задач можно выделить следующие их модели (рис. 1.2):

- распределенное представление (Distribution Presentation);
- удаленное представление (Remote Presentation);
- распределенная бизнес-логика (Remote Business Logic);
- удаленное управление данными (Remote Data Management);
- распределенное управление данными (Distributed Data Management).

Эта условная классификации показывает, как могут быть распределены отдельные задачи между серверным и клиентскими процессами. В данной классификации отсутствует реализация удаленной бизнес-логики, так как считается, что она не может быть удалена полностью, а может быть лишь распределена между разными процессами, которые могут взаимодействовать друг с другом.

1.3. Двухуровневые модели

Двухуровневые модели управления БД фактически являются результатом распределения пяти указанных ранее групп функций стандартного интерактивного приложения между двумя процессами, выполняемыми на двух платформах: компьютере клиента и на сервере. В чистом виде не существует ни одна из них, однако рассмотрим наиболее характерные особенности каждой двухуровневой модели.

Модель удаленного управления данными, или модель файлового сервера (File Server — FS). В этой модели презентационная логика и бизнес-логика располагаются на клиентской части. На сервере располагаются файлы с данными и поддерживается доступ к этим файлам. Функции управления информационными ресурсами в этой модели находятся на клиентской части.

Распределение функций компонентов приложения в моделях клиент—сервер представлено на рис. 1.3.

В этой модели файлы базы данных хранятся на сервере, клиент обращается к серверу с файловыми командами, а механизм управления всеми информационными ресурсами — собственно база метаданных (выбранных данных) — находится на компьютере клиента.

Достоинство данной модели состоит в том, что приложение разделено на два взаимодействующих процесса. При этом сервер

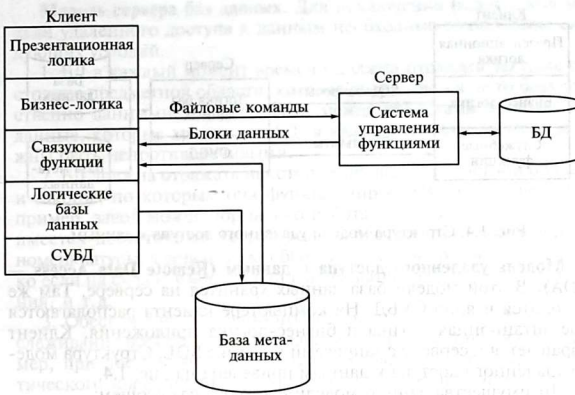


Рис. 1.3. Модель файлового сервера

(серверный процесс) может обслуживать множество клиентов, которые обращаются к нему с запросами. Собственно СУБД должна находиться в этой модели на компьютере клиента.

Алгоритм выполнения клиентского запроса сводится к следующему.

1. Запрос формулируется в командах языка манипулирования данными (ЯМД).

2. СУБД переводит этот запрос в последовательность файловых команд.

3. Каждая файловая команда вызывает перекачку блока информации на компьютер клиента, а СУБД анализирует полученную информацию, и если в полученном блоке не содержится ответ на запрос, принимается решение о перекачке следующего блока информации и т.д.

4. Перекачка информации с сервера на клиентский компьютер производится до тех пор, пока не будет получен ответ на запрос клиента.

Рассмотренная модель имеет следующие недостатки:

- высокий сетевой трафик, который связан с передачей по сети множества блоков и файлов, необходимых приложению;
- узкий спектр операций манипулирования с данными, определяемый только файловыми командами;
- отсутствие адекватных средств безопасности доступа к данным (защита только на уровне файловой системы).

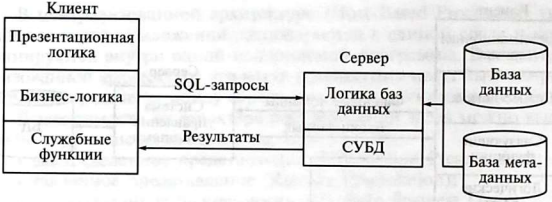


Рис. 1.4. Структура модели удаленного доступа к данным

Модель удаленного доступа к данным (Remote Data Access — RDA). В этой модели база данных хранится на сервере. Там же находится и ядро СУБД. На компьютере клиента располагаются презентационная логика и бизнес-логика приложения. Клиент обращается к серверу с запросами на языке SQL. Структура модели удаленного доступа к данным приведена на рис. 1.4.

Преимущества данной модели состоят в следующем:

- перенос компонента представления и прикладного компонента на клиентский компьютер существенно разгружает сервер БД, сводя к минимуму общее число выполняемых процессов в операционной системе;
- сервер БД освобождается от несвойственных ему функций, а процессор или процессоры сервера целиком загружаются операциями обработки данных запросов и транзакций;
- резко уменьшается загрузка сети, так как по ней от клиентов к серверу передаются не запросы на ввод-вывод в файловой терминологии, а запросы на языке SQL, объем которых существенно меньше. В ответ же на эти запросы клиент получает только данные, соответствующие запросу, а не блоки файлов.

Основным достоинством модели RDA является унификация интерфейса клиент—сервер, т.е. стандартным при общении приложения клиента и сервера становится язык SQL.

Недостатки данной модели:

- запросы на языке SQL при интенсивной работе клиентской части приложения могут существенно загрузить сеть;
- так как в этой модели на компьютере клиента располагаются и презентационная логика, и бизнес-логика приложения, при повторении аналогичных функций в других приложениях код, соответствующей бизнес-логики, должен быть повторен для каждого клиентского приложения, что вызывает излишнее их дублирование;
- так как сервер в этой модели играет пассивную роль, функции управления информационными ресурсами должны выполняться на компьютере клиента.

Модель сервера баз данных. Для исключения недостатков модели удаленного доступа к данным необходимо выполнение следующих условий.

1. БД в каждый момент времени должна отражать текущее состояние предметной области, которое определяется не только собственно данными, но и связями между объектами данных, т.е. данные, которые хранятся в БД, в каждый момент времени должны быть непротиворечивыми.

2. БД должна отражать некоторые правила предметной области и законы, по которым она функционирует (Business Rules). Например, завод может нормально работать, только если на складе имеется достаточный (страховой) запас деталей определенной номенклатуры, а деталь может быть запущена в производство, только если на складе имеется достаточно материала для ее изготовления, и т.д.

3. Обеспечение постоянного контроля за состоянием БД, отслеживание всех изменений и адекватная реакция на них. Например, при достижении некоторым измеряемым параметром критического значения должно произойти отключение определенной аппаратуры, при уменьшении товарного запаса ниже допустимой нормы должна быть сформирована заявка конкретному поставщику на поставку соответствующего товара и т.п.

4. Возникновение некоторой ситуации в БД должно четко и оперативно влиять на ход выполнения прикладной задачи.

5. Совершенствование контроля типов данных СУБД. В настоящее время СУБД контролирует синтаксически только стандартно-допустимые типы данных, т.е. которые определены в DDL (Data Definition Language) — языке описания данных, являющемся частью SQL. Однако в реальных предметных областях существуют данные, которые несут в себе еще и семантическую составляющую, например координаты объектов или единицы измерений.

Модель сервера баз данных поддерживают большинство современных СУБД: Informix, Ingres, Sybase, Oracle, MS SQL Server. Основу данной модели составляют: механизм хранимых процедур как средство программирования SQL-сервера, механизм триггеров как механизм отслеживания текущего состояния информационного хранилища и механизм ограничений на пользовательские типы данных, который иногда называют механизмом поддержки доменной структуры.

Модель активного сервера базы данных представлена на рис. 1.5. В этой модели бизнес-логика разделена между клиентом и сервером. На сервере бизнес-логика реализована в виде хранимых процедур — специальных программных модулей, которые хранятся в БД и управляются непосредственно СУБД. Клиентское приложение обращается к серверу с командой запуска хранимой процедуры, а сервер выполняет эту процедуру и регистрирует все предус-

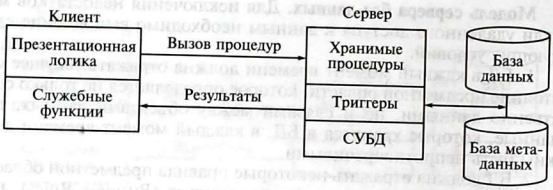


Рис. 1.5. Модель активного сервера базы данных

мотренные изменения в БД. Сервер возвращает клиенту данные выполненного запроса, которые требуются клиенту либо для вывода на экран, либо для выполнения части бизнес-логики. При этом трафик обмена информацией между клиентом и сервером резко уменьшается.

Централизованный контроль в модели сервера баз данных выполняется с использованием механизма триггеров, которые также являются частью БД.

Термин «триггер», взятый из электроники, семантически очень точно характеризует механизм отслеживания специальных событий, связанных с состоянием БД. Триггер является как бы некоторым тумблером, который срабатывает при возникновении определенного события в БД. Ядро СУБД проводит мониторинг всех событий, вызывающих созданные и описанные триггеры в БД, и при возникновении такого события сервер запускает соответствующий триггер. Каждый триггер представляет собой также некоторую программу, которая выполняется с базой данных. С помощью триггеров можно вызывать хранимые процедуры.

Механизм использования триггеров предполагает, что при срабатывании одного из них могут возникнуть события, которые вызовут срабатывание других.

В данной модели сервер является активным, так как в ней не только клиент, но и сам сервер, используя механизм триггеров, может быть инициатором обработки данных в БД.

Хранимые процедуры и триггеры хранятся в словаре БД и, следовательно, могут быть использованы несколькими клиентами, что существенно уменьшает дублирование алгоритмов обработки данных в разных клиентских приложениях.

Недостатком данной модели является очень большая загрузка сервера, так как он обслуживает множество клиентов и выполняет следующие функции:

• осуществляет мониторинг событий, связанных с выполнением разработанных триггеров;

• обеспечивает автоматическое срабатывание триггеров при возникновении связанных с ними событий;
• обеспечивает исполнение внутренней программы каждого триггера;
• запускает хранимые процедуры по запросам пользователей;
• запускает хранимые процедуры из триггеров;
• возвращает требуемые данные клиенту;
• обеспечивает выполнение всех функций СУБД (доступ к данным, контроль и поддержку целостности данных в БД, контроль доступа, обеспечение корректной параллельной работы всех пользователей с единой БД).

Если перенести на сервер бóльшую часть бизнес-логики приложений, то требования к клиентам в этой модели резко уменьшатся. Иногда такую модель называют моделью с тонким клиентом, а рассмотренные ранее модели — моделями с толстым клиентом.

Модель сервера приложений. Эта трехуровневая модель, являющаяся расширением двухуровневой модели, т.е. с введенным дополнительным промежуточным уровнем между клиентом и сервером, была предложена для разгрузки сервера.

Архитектура трехуровневой модели приведена на рис. 1.6. Промежуточный уровень может содержать один или несколько серверов приложений.

В данной модели компоненты приложения делятся между тремя исполнителями: клиентом, сервером приложений и сервером базы данных.

Клиент обеспечивает логику представления, включая графический пользовательский интерфейс и локальные редакторы. Клиент может запускать локальный код приложения клиента, который может содержать обращения к локальной БД, расположенной на его компьютере. Клиент исполняет коммуникационные функции front-end части приложения, обеспечивающие ему доступ в локальную или глобальную сеть. Дополнительно реализация взаимодействия между клиентом и сервером может включать в себя управление распределенными транзакциями, что соответствует случаям, когда клиент также является клиентом менеджера распределенных транзакций.

Рис. 1.6. Модель сервера приложений

Серверы приложений, составляющие новый промежуточный уровень архитектуры модели, спроектированы для исполнения общих не загружаемых функций клиентов. Серверы приложений поддерживают функции клиентов как частей взаимодействующих рабочих групп, сетевую доменную операционную среду и каталоги с данными, а также хранят и исполняют наиболее общие правила бизнес-логики, обеспечивают обмен сообщениями и поддержку запросов (особенно в распределенных транзакциях).

Серверы баз данных в этой модели занимаются исключительно функциями СУБД: обеспечивают функции создания и ведения БД, поддерживают целостность реляционной БД, обеспечивают функции хранилищ данных (warehouse services). Кроме того, на них возлагаются функции создания резервных копий БД и восстановления БД после сбоев, управления выполнением транзакций и поддержки устаревших (унаследованных) приложений (legacy application).

Данная модель обладает большей гибкостью, чем двухуровневые модели. Наиболее заметны преимущества модели сервера приложений в тех случаях, когда выполняются сложные аналитические расчеты в базе данных, относящихся к области OLAP-приложений (On-line analytical processing). В этой модели бóльшая часть бизнес-логики клиента изолирована от возможностей встроенного языка SQL, реализованного в конкретной СУБД, и может быть выполнена на языках программирования C, C++, Sir.i. Talk, Cobol, что повышает переносимость системы и ее масштабируемость.

Модели серверов баз данных. При создании первых СУБД технология клиент—сервер только зарождалась, поэтому изначально в архитектуре этих систем не было адекватного механизма организации взаимодействия клиентского и серверного процессов. В современных СУБД этот механизм является фактически основополагающим и от эффективности его реализации зависит эффективность работы системы в целом.

Рассмотрим эволюцию подобных механизмов. В основном такой механизм определяется структурой реализации серверных процессов и часто называется *архитектурой сервера баз данных.*

Как уже отмечалось, поначалу существовала модель, в которой управление данными (функция сервера) и взаимодействие с пользователем были совмещены в одной программе. Этот этап развития серверов БД можно назвать нулевым.

Затем функции управления данными были выделены в самостоятельную группу — сервер. Однако при этом модель взаимодействия пользователя с сервером соответствовала структуре связей между таблицами баз данных «один к одному» (рис. 1.7), т.е. сервер обслуживал запросы только одного пользователя (клиента), а для обслуживания нескольких клиентов нужно было запустить соответствующее число серверов.

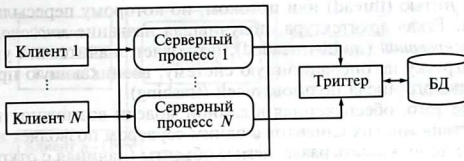


Рис. 1.7. Взаимодействие клиентских и серверных процессов в модели «один к одному»

Выделение сервера в отдельную программу было революционным шагом, который позволил, в частности, поместить сервер на одну машину, а программный интерфейс с пользователем — на другую и осуществлять взаимодействие между ними по сети. Однако необходимость запуска большого числа серверов для обслуживания множества пользователей сильно ограничивала возможности такой системы. Для обслуживания большого числа клиентов на сервере следовало одновременно запустить в работу большое число серверных процессов, что резко повышало требования к ресурсам ЭВМ.

Кроме того, каждый серверный процесс в этой модели запускался как независимый, поэтому запрос, который только что был выполнен одним серверным процессом, при формировании его другим клиентом выполнялся повторно. В такой модели сложно обеспечить взаимодействие серверных процессов. Эта модель серверов баз данных самая простая, и исторически она появилась первой.

Проблемы, возникающие в информационной модели «один к одному», решены в архитектуре систем с выделенным сервером, который способен обрабатывать запросы от многих клиентов. В этой системе сервер единственный обладает монополией на управление данными и взаимодействует одновременно со многими клиентами (рис. 1.8). Логически каждый клиент связан с сервером от-

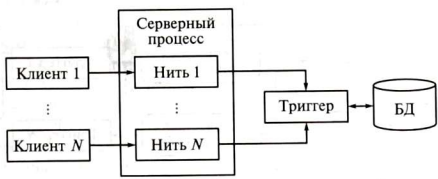


Рис. 1.8. Многопотоковая односерверная архитектура

дельной нитью (thread) или потоком, по которому пересылаются запросы. Такая архитектура, получившая название *многопотоковой односерверной* (multi-threaded), позволяет значительно уменьшить нагрузку на операционную систему, возникающую при работе большого числа пользователей (trashing).

Кроме того, обеспеченная в данной модели возможность взаимодействия многих клиентов с одним сервером позволяет в полной мере использовать разделяемые объекты (начиная с открытых файлов и заканчивая данными из системных каталогов), что значительно уменьшает потребность в памяти и общее число процессов операционной системы. Например, в модели «один к одному» СУБД создает 100 копий процессов для 100 пользователей, а системе с многопотоковой архитектурой для этого понадобится только один серверный процесс.

Однако такое решение имеет свои недостатки. Так как сервер может выполняться только на одном процессоре, возникает естественное ограничение на применение СУБД для мультипроцессорных платформ. Если компьютер имеет, например, четыре процессора, то СУБД с одним сервером использует только один из них, не загружая другие три.

В некоторых системах эта проблема решается вводом промежуточного диспетчера, и созданная таким образом система называется *архитектурой с виртуальным сервером* (virtual server). В такой архитектуре (рис. 1.9) клиенты подключаются не к реальному серверу, а к промежуточному звену, называемому диспетчером и выполняющему только функции диспетчеризации запросов к серверам. В этом случае нет ограничений на использование многопроцессорных платформ и число серверов может быть согласовано с числом процессоров в системе.

Однако и такая архитектура не лишена недостатков, так как здесь в систему добавляется новый слой, размещаемый между клиентом и сервером, что увеличивает потребность в ресурсах на поддержку баланса загрузки серверов (load balancing) и ограничивает возможности управления взаимодействием между клиентом и сервером. Во-первых, становится невозможно направить запрос от конкретного клиента конкретному серверу; во-вторых,

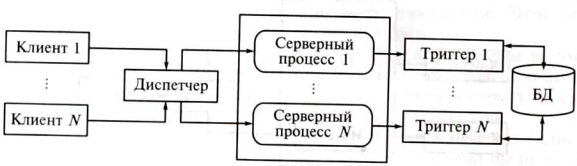


Рис. 1.9. Архитектура с виртуальным сервером

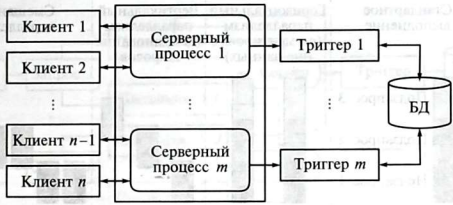


Рис. 1.10. Многопотоковая мультисерверная архитектура

серверы становятся равноправными, так как нет возможности устанавливать приоритеты для обслуживания запросов.

Подобная организация взаимодействия между клиентом и сервером аналогична организации процесса обслуживания в банке, где имеются несколько окон кассиров и специальный банковский служащий — администратор зала, направляющий каждого пришедшего посетителя (клиента) к свободному кассиру (актуальному серверу). Данная система работает нормально, пока все посетители равноправны (имеют равные приоритеты), однако стоит появиться посетителям, которые должны обслуживаться в специальном окне, возникают проблемы. Учет приоритета клиентов особенно важен в системах оперативной обработки транзакций, однако именно эту возможность не может предоставить архитектура систем с диспетчеризацией.

Современное решение проблемы СУБД для мультипроцессорных платформ заключается в возможности запуска нескольких серверов базы данных, в том числе и на различных процессорах. При этом каждый из серверов должен быть многопотоковым. Если эти два условия выполнены, есть основания говорить о многопотоковой архитектуре с несколькими серверами, показанной на рис. 1.10, которую также называют *многонитиевой мультисерверной архитектурой*.

Эта архитектура обеспечивает распараллеливание выполнения одного пользовательского запроса несколькими серверными процессами, т.е. пользовательский запрос разбивается на ряд подзапросов, которые могут выполняться параллельно, а результаты их выполнения потом объединяются в общий результат выполнения запроса. Следовательно, в этом случае для обеспечения оперативности выполнения запросов их подзапросы могут быть направлены отдельным серверным процессам, а затем полученные результаты объединены в общий результат (см. рис. 1.10). Серверные процессы в данном случае не являются независимыми и их принято называть *нитями* (treads). Управление нитями множества запросов пользователей требует дополнительных расходов от

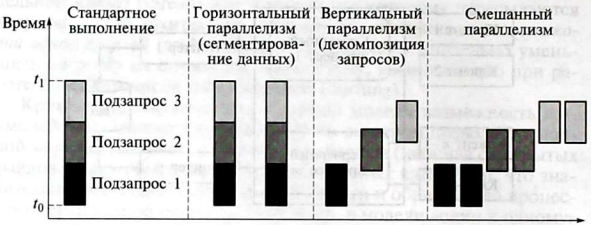


Рис. 1.11. Типы параллелизма

СУБД, однако при оперативной обработке информации в хранилищах данных такой подход наиболее перспективен.

Типы параллелизма. Рассматривают следующие программно-аппаратные способы распараллеливания запросов: горизонтальный, вертикальный и смешанный (рис. 1.11).

Горизонтальный параллелизм возникает, когда хранимая в БД информация распределяется по нескольким физическим устройствам хранения, т.е. нескольким дискам. При этом информация из одного отношения разбивается на части по горизонтали.

Горизонтальный параллелизм иногда называют распараллеливанием, или сегментированием, данных. Параллельность здесь достигается выполнением одинаковых операций, например фильтрации, над разными хранимыми физическими данными. Эти операции могут выполняться параллельно разными процессами, так как они независимы. Результат выполнения целого запроса складывается из результатов выполнения отдельных операций.

Время выполнения запроса при соответствующем сегментировании данных существенно меньше, чем время выполнения этого же запроса традиционными способами одним процессом.

Вертикальный параллелизм достигается конвейерным выполнением операций, составляющих запрос пользователя. Такой под-

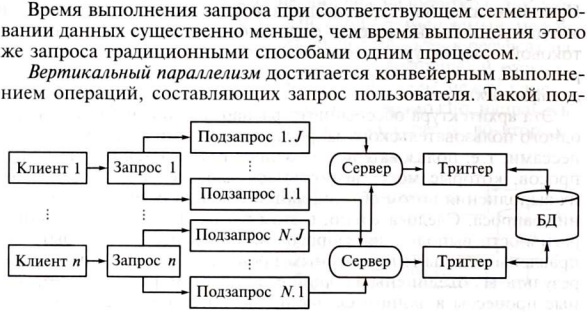


Рис. 1.12. Схема выполнения запроса при вертикальном параллелизме

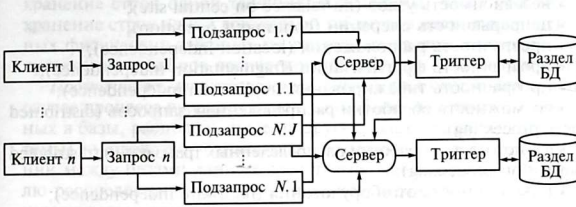


Рис. 1.13. Схема выполнения запроса при смешанном параллелизме

ход, требующий серьезного усложнения модели выполнения реляционных операций ядром СУБД, предполагает, что ядро может производить декомпозицию запроса, базируясь на его функциональных компонентах, и при этом ряд подзапросов может выполняться параллельно с минимальной связью между отдельными шагами выполнения запроса.

Действительно, рассмотрев для примера следующую последовательность операций реляционной алгебры:

1. $R5 = R1\ [A, C]$;
2. $R6 = R2\ [A, B, D]$;
3. $R7 = R5\ [A > 128]$;
4. $R8 = R5\ [A] R6$,

увидим, что первую и третью операции можно объединить и выполнить параллельно со второй операцией, а затем с полученными результатами выполнить последнюю — четвертую — операцию.

Общее время выполнения подобного запроса, конечно, будет существенно меньше, чем при традиционном способе выполнения последовательности из четырех операций (рис. 1.12).

Смешанный параллелизм является гибридом горизонтального и вертикального (рис. 1.13).

Все виды параллелизма применяются в приложениях, где они позволяют существенно сократить время выполнения сложных запросов из нескольких таблиц с большим объемом данных.

1.4. Основные свойства распределенных баз данных

С точки зрения пользователей распределенная база данных выглядит как обычная настольная база данных, компоненты которой могут находиться на различных компьютерах (узлах) локальной сети предприятия.

В идеале для распределенных баз данных должны быть характерны следующие свойства:

- локальная автономия (local autonomy);

• независимость узлов (no reliance on central site);
• непрерывность операций (continuous operation);
• прозрачность расположения (location independence);
• прозрачность фрагментации (fragmentation independence);
• прозрачность тиражирования (replication independence);
• возможность обработки распределенных запросов (distributed query processing);
• возможность обработки распределенных транзакций (distributed transaction processing);
• независимость от оборудования (hardware independence);
• независимость от операционных систем (operationg system independence);
• прозрачность сети (network independence);
• независимость от баз данных (database independence).

Локальная автономия — свойство, означающее, что управление данными на каждом из узлов распределенной системы выполняется локально. База данных, расположенная на одном из узлов, является неотъемлемым компонентом распределенной системы. Будучи фрагментом общего пространства данных, она в то же время функционирует как полноценная локальная база данных, управление которой выполняется локально и независимо от других узлов системы.

Независимость узлов — свойство, означающее, что в идеальной системе все узлы равноправны и независимы, а расположенные на них базы являются равноправными поставщиками данных в общее пространство данных. База данных на каждом из узлов самодостаточна, т.е. она включает в себя полный собственный словарь данных и полностью защищена от несанкционированного доступа.

Непрерывность операций — свойство, которое можно трактовать как возможность непрерывного доступа к данным (24 ч в сутки или семь дней в неделю) в рамках DDB независимо от их расположения и независимо от операций, выполняемых на локальных узлах. Это свойство можно выразить следующим образом: данные доступны всегда, а операции над ними выполняются непрерывно.

Прозрачность расположения — свойство, означающее полную прозрачность расположения данных. Пользователь, обращающийся к DDB, ничего не должен знать о реальном (физическом) размещении данных в узлах информационной системы. Все операции с данными выполняются без учета их местонахождения. Транспортировка запросов к базам данных осуществляется встроенными системными средствами.

Прозрачность фрагментации — свойство, которое трактуется как возможность распределенного размещения данных, логически представляющих собой единое целое. Существует фрагментация двух типов: горизонтальная и вертикальная. Первая означает

хранение строк одной таблицы на различных узлах (фактически хранение строк одной логической таблицы в нескольких идентичных физических таблицах на различных узлах), а вторая — распределение столбцов логической таблицы по нескольким узлам.

Прозрачность тиражирования данных (асинхронного в общем случае процесса переноса изменений объектов исходной базы данных в базы, расположенные на других узлах распределенной системы) — свойство, означающее возможность переноса изменений между базами данных средствами, невидимыми пользователю распределенной системы, или, что тиражирование возможно и достигается внутрисистемными средствами.

Возможность обработки распределенных запросов — свойство DDB, которое трактуется как возможность выполнения операций выборки информации из распределенной базы данных, сформулированных в рамках обычного запроса на языке SQL. Это означает, что операцию выборки из DDB можно сформулировать с помощью тех же языковых средств, что и операцию в локальной базе данных.

Возможность обработки распределенных транзакций — свойство DDB, которое можно трактовать как возможность выполнения операций обновления распределенной базы данных (INSERT, UPDATE, DELETE), не разрушая целостность и согласованность данных, что достигается применением двухфазового (или двухфазного) протокола фиксации транзакций (two-phase commit protocol), ставшего фактическим стандартом обработки распределенных транзакций. Использование этого протокола гарантирует согласованное изменение данных на нескольких узлах в рамках распределенной (или глобальной) транзакции.

Независимость от оборудования — свойство, означающее, что в качестве узлов распределенной системы могут выступать компьютеры любых моделей и производителей.

Независимость от операционных систем — свойство, вытекающее из предыдущего свойства и означающее многообразие операционных систем, управляющих узлами распределенной системы.

Прозрачность сети — свойство, означающее, что в распределенной системе возможны любые сетевые протоколы, т.е. доступ к любым базам данных может осуществляться по сети и спектр поддерживаемых конкретной СУБД сетевых протоколов не должен быть ограничением системы с распределенными базами данных.

Независимость от баз данных — свойство, означающее, что в распределенной системе могут сосуществовать СУБД различных производителей, а также возможны операции поиска и обновления в базах данных различных моделей и форматов.

Рассмотренные свойства во многом связаны с технологиями, осуществляющими доступ и обработку информации в удаленных базах данных.

Контрольные вопросы

1. Дайте определения следующих терминов: топология БД (или структура распределенной БД), локальная автономность, удаленный запрос, поддержка распределенной транзакции, презентационная логика, бизнес-логика.

2. Какие двухуровневые модели вы знаете? Назовите их достоинства и недостатки.

3. Назовите характеристики следующих архитектур организации баз данных: многопотоковая односерверная архитектура, архитектура с виртуальным сервером, многонитиевая мультисерверная архитектура.

4. Для чего применяют распараллеливание запросов и какие типы параллелизма вы знаете?

ГЛАВА 2

ПРИНЦИПЫ РАЗРАБОТКИ И ЭКСПЛУАТАЦИИ СИСТЕМ УПРАВЛЕНИЯ УДАЛЕННЫМИ БАЗАМИ ДАННЫХ

2.1. CALS-технологии — основная концепция разработки удаленных баз данных

Как уже говорилось, CALS-технологии — это современное направление развития информационного обеспечения производственных и бизнес-процессов, направленное на создание единого информационного пространства, основу которого составляют удаленные интегрированные базы данных.

Концепция и идеология CALS зародилась в недрах военно-промышленного комплекса США, а затем была принята всеми странами НАТО.

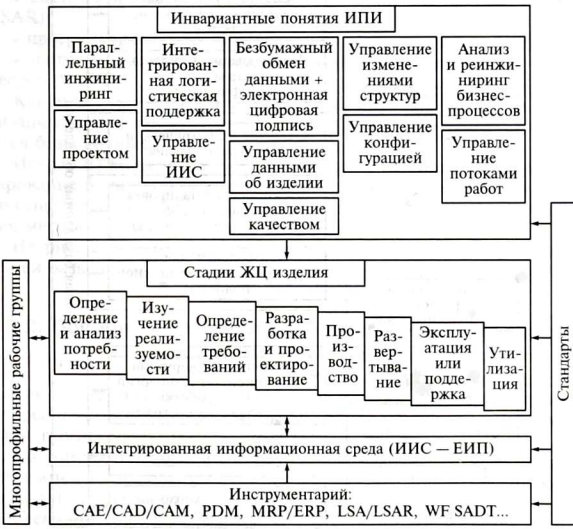


Рис. 2.1. Схема ИПИ

В России принят адекватный аналог CALS — информационная поддержка жизненного цикла изделий (ИПИ). На рис. 2.1 показана схема, отражающая суть ИПИ.

В соответствии с данной схемой основу ИПИ составляет интегрированная информационная среда (ИИС), или единое информационное пространство (ЕИП). Эти термины равнозначны, однако в терминологическом словаре, утвержденном Госстандартом России, принят первый термин — ИИС.

Стандарт определяет ИИС как совокупность удаленных распределенных баз данных, содержащих сведения об изделиях, производственной среде, ресурсах и процессах предприятия, обеспечивающих корректность, актуальность, сохранность и доступность данных тем субъектам производственно-хозяйственной деятель-

Рис. 2.2. Компоненты общей базы данных промышленного предприятия

ности, участвующим в осуществлении жизненного цикла изделия, которым это необходимо и разрешено.

При создании на предприятии ИИС должен реализовываться главный принцип ИПИ: информация, однажды возникшая на каком-либо этапе производственного процесса, сохраняется и становится доступной всем участникам этого или других этапов в соответствии с имеющимися у них правами пользования данной информацией.

Естественно, что процессы создания, преобразования и передачи информации осуществляются с помощью современных программных средств, к числу которых, как показано на рис. 2.1, относятся:

- системы автоматизированного конструкторского и технологического проектирования (CAE/CAD/CAM);
- программные средства управления данными об изделиях, в том числе СУБД (PDM);
- автоматизированные системы планирования и управления производством (MRP/ERP);
- системы анализа, поддержки и ведения баз данных (LSA/LSAR);
- программные средства управления потоками работ (WF);
- программные средства моделирования и анализа бизнес-процессов (SADT).

Как видно из перечисленных задач и применяемых для их реализации программных систем, основой для создания ИИС являются базы данных.

Исходя из концепции CALS-технологий традиционное проектирование базы данных как самостоятельного объекта необходимо существенным образом изменить и перейти к стратегии создания многопользовательских — общих — баз данных.

На рис. 2.2 показаны некоторые компоненты формирования общей базы данных промышленного предприятия.

2.2. Принципы разработки многопользовательских информационных систем

Как следует из концепции CALS-технологий, разрабатываемые на предприятиях информационные системы и базы данных должны быть многопользовательскими.

Принципы разработки многопользовательских баз данных заключаются в соблюдении двух обязательных условий: системный подход и стандартизация.

Системный подход к разработке информационной системы означает, что такая система рассматривается как «большая система», состоящая из некоторого множества взаимосвязанных и вза-

имодействующих между собой элементов. При проектировании информационных систем необходимо:

- учитывать интересы всех потенциальных пользователей систем;
- использовать модульный принцип разработки и внедрения.

Принцип учета интересов всех потенциальных пользователей системы определяет следующий порядок разработки БД.

1. Установить, каким специалистам и в каких подразделениях предприятия необходима информация о конкретном информационном объекте.
2. Установить признаки описания объектов различными пользователями.
3. Установить общий состав признаков объектов одного класса.

Такой подход к проектированию увеличивает сроки разработки БД, но обеспечивает значительное снижение затрат на разработку всей системы в целом.

Для пояснения данного принципа приведем реальный пример разработки БД на одном из предприятий, где появление программ создания баз данных было по достоинству оценено сотрудниками, и они стали разрабатывать необходимые для себя базы данных.

Так как одной из задач, стоящих перед технологами цехов, являлся выбор инструмента для механической обработки деталей, они разработали свою цеховую БД по режущему инструменту (затратив на это и время и средства).

В то же время в конструкторском отделе завода специалисты, занимающиеся проектированием режущего инструмента, также создали свою БД. Однако когда руководство приняло решение создать общезаводскую информационную систему по режущему инструменту, оказалось, что одни и те же признаки режущего инструмента разные специалисты описывали разными способами. В результате разработанные базы данных пришлось полностью переделывать, что потребовало как дополнительного времени, так и дополнительных затрат. Средства, затраченные на разработку несогласованных между специалистами баз данных, были потеряны для предприятия.

Модульный принцип разработки и внедрения БД означает, что любая система должна разрабатываться в виде отдельных взаимосвязанных модулей (подсистем), которые могут внедряться в производство отдельно, т.е. до окончательной разработки всей системы.

Стандартизация разработки информационных систем, учитывая их многопользовательский характер, включает в себя следующие аспекты: информационный, программный и аппаратный.

Стандартизация информационного обеспечения обусловлена принципами компьютерной обработки информации, при которой объекты баз данных должны однозначно распознаваться компьютером.

Применительно к текстовой информации этот аспект разработки БД означает, что четкие правила идентификации (грамматические правила написания) должны быть установлены для всех информационных объектов. Так, установив название инструмента для механической обработки детали *резец расточной*, недопустимо использовать никакой другой способ его обозначения, т.е. название *расточной резец* не идентично названию *резец расточной*.

Стандартизация программного обеспечения необходима, так как при разработке многопользовательских, удаленных друг от друга систем данные одной системы должны обрабатываться программным обеспечением другой системы.

Стандартизация аппаратного обеспечения обусловлена необходимостью снижения затрат на эксплуатацию компьютерной техники.

Внедрение в настоящее время на предприятиях России концепции CALS-технологий предусматривает широкое применение единых, в том числе и международных, стандартов.

2.3. Организация многопользовательских систем управления базами данных в локальных вычислительных сетях

Компьютерные информационные системы современных предприятий разрабатываются с применением сетевых технологий, т.е. компьютеры объединяют в локальные вычислительные сети. При разработке баз данных в ЛВС предприятий применяют два типа (две архитектуры) их организации: файл—сервер и клиент—сервер.

Общими признаками для этих типов организации баз данных является наличие *сервера* (компьютера), на котором находятся базы (файлы) данных, и *рабочих станций* (компьютеров пользователей) — *клиентов*.

Отличаются эти две архитектуры организации баз данных способами обработки информации.

В архитектуре файл—сервер все процессы обработки информации производятся на компьютере клиента, для чего ему по соответствующему запросу пересылается весь файл с данными.

В архитектуре клиент—сервер все процессы обработки информации выполняются на сервере по запросу клиента, которому отсылаются только результаты обработки данных.

При организации многопользовательских сетевых баз данных предпочтительной является организация их по типу клиент—сервер, что обусловлено следующими недостатками архитектуры файл—сервер и преимуществами архитектуры клиент—сервер.

Недостатки организации БД по архитектуре файл—сервер:

• при передаче по сети файлов БД (особенно с большими объемами информации и с учетом возможного обращения к файлам одновременно нескольких пользователей) резко снижается производительность работы с системой;

• при одновременной передаче по сети файлов с большими объемами нескольким пользователям увеличивается вероятность нарушения достоверности передаваемой информации, т.е. снижается надежность работы системы.

Преимущества организации БД по архитектуре клиент—сервер:

• при передаче по сети только результатов обработки данных по запросам клиентов резко снижается нагрузка на сеть, а следовательно, увеличивается возможность подключения к БД большего числа пользователей), т.е. производительность данной системы значительно выше, чем в архитектуре файл—сервер;

• централизованное хранение и обработка данных на сервере повышает надежность работы системы;

• разработку серверной части СУБД можно выполнять на языке SQL или на других языках высокого уровня, что повышает надежность и производительность обработки данных. Разработку клиентской части СУБД можно выполнять с применением прикладных программных продуктов, например Visual Basic и Microsoft Access, что значительно сокращает время разработки информационной системы.

2.4. Этапы проектирования многопользовательских баз данных

Установлено, что в современных условиях развития производства и бизнеса необходимо переходить от стратегии проектирования баз данных как самостоятельных объектов на стратегию создания многопользовательских информационных систем, т.е. общих баз данных.

Такой переход предусматривает необходимость разработки СУБД данных в соответствии с этапами их жизненного цикла, содержание которых представлено в табл. 2.1.

Рассмотрим более подробно работы, выполняемые на каждом этапе жизненного цикла удаленных баз данных.

Планирование разработки базы данных — это подготовительные работы, включающие в себя нахождение методов и средств оптимального решения задачи, поставленной перед информационной системой.

Планирование разработки базы данных должно быть неразрывно связано с общей стратегией формирования единого информационного пространства предприятия, для чего необходимо:

Таблица 2.1

Этапы жизненного цикла СУБД

Этап	Выполняемые работы
1. Планирование разработки базы данных	Формулирование цели создания базы данных, поиск и обоснование оптимальных методов (способов) организации СУБД в условиях конкретного предприятия
2. Определение требований к СУБД	Определение состава пользователей и разграничение задач между ними в процессе проектирования и эксплуатации СУБД
3. Разработка единого описания характеристик объекта базы данных	Сбор и анализ требований к описанию объектов базы данных всех потенциальных пользователей информационной системы
4. Разработка и исследование моделей проекта СУБД	Концептуальное, логическое и физическое моделирование базы данных
5. Обоснование и выбор программной системы для разработки баз данных	Оценка ожидаемых затрат на разработку и эксплуатацию базы данных в условиях предприятия
6. Разработка «эскизного проекта» — прототипа базы данных (этап необязательный)	Создание модели базы данных средствами визуального проектирования, например Microsoft Access
7. Разработка приложения	*Серверная часть СУБД*: разработка объектов базы данных (структуры таблиц базы данных и связей между ними, обеспечивающих целостность данных) и физическая реализация таблиц БД в конкретной СУБД *Клиентская часть СУБД*: разработка запросов и отчетов в соответствии с решаемыми задачами, сценария и форм пользовательского интерфейса, программ управления и доступа к удаленным базам данных
8. Реализация СУБД	Создание внешнего концептуального и внутреннего определений базы данных и прикладных программ
9. Загрузка данных	Заполнение информацией таблиц баз данных

Окончание табл. 2.1

Этап	Выполняемые работы
10. Тестирование системы	Проверка работы баз данных и устранение возникающих ошибок в работе приложения
11. Эксплуатация и сопровождение системы	Разработка организационных мероприятий по внедрению системы, постоянное наблюдение за ее работой и при необходимости внесение изменений в разработанное приложение

- определить цель и задачи информационных технологий на основе анализа целей и бизнес-планов организации;
- провести анализ существующих (в том числе действующих на предприятии) информационных систем и дать рекомендации по их применению или модернизации;
- дать оценку экономической эффективности, ожидаемой от разработки новых информационных технологий (в том числе ожидаемых преимуществ перед конкурентами).

Очевидно, что целью разработки любой компьютерной системы является достижение определенного экономического эффекта от ее реализации. Следовательно, в условиях конкретного предприятия необходимо установить приоритетные направления в создании баз данных. Базы данных могут разрабатываться практически для всех задач управления производством, например:

- поставка материалов и комплектующих изделий;
- проектирование конструкций новых изделий;
- проектирование технологических процессов изготовления продукции;
- проектирование технологического оснащения (приспособлений, инструмента);
- оперативное календарное планирование и управление выпуском изделий;
- разработка нормативной базы (потребности в трудовых и материальных ресурсах, основных и вспомогательных материалах и др.);
- управление качеством выпускаемой продукции;
- управление сбытом.

Принятие решения о выборе направления для разработки баз данных является прерогативой руководителей предприятия.

В результате выполнения работ данного этапа ЖЦ базы данных получают:

- техническое задание на проектирование;
- технические требования на методы и средства выполнения конкретных задач при работе с базой данных;
- стандарты предприятия на выполнение отдельных этапов работ;

• необходимые трудовые и материальные ресурсы для разработки и эксплуатации баз данных;

• планируемые сроки выполнения 2...10 этапов жизненного цикла баз данных.

Определение требований к СУБД зависит от области применения баз данных, состава пользователей, а следовательно, и от назначения системы.

Выбрав область производственной деятельности, необходимо установить состав пользователей информацией разрабатываемой базы данных. Это требуется для решения следующих задач:

• определение классов информационных объектов, их характеристик и в конечном счете состава таблиц баз данных;

• определение места нахождения потенциальных пользователей и в конечном счете архитектуры ЛВС.

Определив состав пользователей баз данных, следует установить задачи для каждого пользователя системы: одним — дать право модифицировать таблицы баз данных, а другим — разрешить только доступ к информации без права ее изменения.

Разработка единого описания характеристик объекта базы данных представляет собой достаточно трудоемкий процесс сбора и анализа информации от каждого потенциального пользователя базой данных.

Существуют разные методы сбора информации, которые в общем определяются как методы сбора фактов. К этим методам относятся:

• изучение документации;

• проведение собеседований;

• наблюдение за работой сотрудников подразделений предприятия;

• проведение исследований;

• проведение анкетирования.

Изучение документации, т.е. определение характеристик информационных объектов на основе технической документации, в соответствии с которой выполняет свои функции конкретный пользователь (подразделение) предприятия. Приведем некоторые виды документов, подлежащих изучению:

• бланки и формы отчетности (бумажные и электронные);

• стандарты предприятия;

• технические характеристики объектов;

• технические и технологические инструкции.

Проведение собеседований — достаточно эффективный метод сбора фактов. При проведении собеседований можно также установить степень заинтересованности пользователей, собрать предложения по организации работ с информационной системой и др.

Наблюдение за работой сотрудников подразделений предприятия относится также к эффективной методике сбора фактов. Данная методика позволяет:

- убедиться в правильности установленных ранее характеристик информационных объектов;
- оценить ожидаемую эффективность от внедрения базы данных в конкретном подразделении благодаря возможности непосредственной оценки трудоемкости выполняемых работ потенциальным пользователем системы.

Проведение исследований на основе изучения технической литературы, ресурсов сети Интернет, материалов конференций и т. п. Достоинство данного метода сводится к возможности изучения методов решения аналогичных проблем другими предприятиями, в том числе и конкурентами.

Проведение анкетирования — метод, основанный на проведении опросов пользователей по заранее составленным опросным листам — анкетам. При этом возможны две формы опросных листов: произвольная и фиксированная. В первом случае опросный лист состоит из вопросов, на которые опрашиваемый (респондент) должен дать ответ в произвольной форме. Во втором случае опрашиваемому предоставляется бланк с вариантами заранее сформулированных ответов на поставленные вопросы, из которых следует сделать выбор.

К данному методу можно также отнести непосредственное «конструирование» таблицы базы данных для задач, выполняемых конкретным специалистом. В этом случае анкетирование можно проводить в виде собеседования или предоставить специалисту самостоятельно составить структуру таблицы (или таблиц) базы данных, для чего целесообразно использовать конструктор таблиц СУБД Microsoft Access.

Разработка и исследование моделей проекта СУБД сводится к разработке и исследованию концептуальных, логических и физических моделей баз данных.

Концептуальное моделирование — это процесс создания информационной модели (базы данных), не зависящей от ее физической реализации. В общем случае это определение необходимого состава таблиц базы данных исходя из установленного состава пользователей.

Логическое моделирование предполагает разработку и установление связей между таблицами базы данных, а также их модификацию на основе принципов нормализации. При разработке логической модели учитывается конкретная СУБД.

Физическое моделирование — это описание способов хранения базы данных на запоминающих устройствах. Очевидно, что данный этап проектирования выполняется после разработки концептуальных и логических моделей баз данных.

Рис. 2.3. Схема моделирования проекта СУБД

Физическое моделирование подразумевает:

- определение конкретных структур хранения информации и методов доступа к удаленным таблицам баз данных;
- выбор аппаратных, программных (технологических) и разработку организационных методов защиты данных.

На рис. 2.3 показана схема моделирования проекта СУБД, которая отражает трехуровневую архитектуру построения и управления базами данных. Данная схема применяется в таких СУБД, как ORACLE и SQL Server. При такой схеме проектирования удаленных баз данных обеспечивается высокая степень независимости системы управления от данных. Причем различают два типа независимости: логическую и физическую.

Логическая независимость от данных означает полную защищенность внешних схем от изменений, вносимых в концептуальную модель.

Физическая независимость от данных означает защищенность концептуальной модели от изменений, вносимых во внутреннюю схему баз данных.

Внешняя схема данных (или внешний уровень) описывает только ту часть управления базами данных, которая относится к каждому пользователю.

Внутренняя схема данных (или внутренний уровень) описывает способы хранения данных.

Обоснование и выбор программной системы для разработки баз данных — это этап, на котором решается задача выбора такой программной системы, которая обеспечивала бы *минимальные трудовые и материальные затраты* при разработке и эксплуатации информационной системы, в частности:

- при разработке баз данных в конкретной предметной области;
- интеграции с уже имеющимися на предприятии базами данных;

• обращении к удаленным базам данных;
• обеспечении защиты данных.

Набольшее распространение среди пользователей и разработчиков СУБД получили следующие программные продукты:
• специальные языки программирования — Visual FoxPro, SQL;
• прикладные программные системы — Microsoft Access;
• программные системы разработки и управления корпоративными удаленными базами данных — Oracle, MS SQL-Server, MYSQL, INFORMIX и др.

На основании имеющегося на 2005 г. опыта можно дать следующие рекомендации по выбору программных систем разработки и управления базами данных.

1. Для разработки единого информационного пространства в рамках CALS-технологий с учетом ожидаемых объемов хранения и обработки данных использовать системы SQL-Server и Oracle различных модификаций.

2. Для разработки и исследования моделей, а также для разработки «эскизного проекта» баз данных использовать Microsoft Access, которая позволит разработчикам в кратчайшие сроки и с наименьшими затратами спроектировать основные объекты баз данных: таблицы, запросы, отчеты. Также, используя возможности Microsoft Access, можно эффективно проработать сценарий будущего пользовательского интерфейса.

3. Для разработки быстродействующего приложения использовать языки объектно-ориентированного программирования: Visual Basic .NET, Delphi и др.

Разработка «эскизного проекта» — прототипа баз данных позволяет проверить разработанные информационные модели баз данных, на основе которых можно с уверенностью составить техническое задание программистам для разработки приложения.

Данный этап должен выполняться либо непосредственно специалистами конкретной предметной области, либо под их руководством. Именно поэтому для реализации данного этапа рекомендуется применять Microsoft Access.

Разработка приложения — это этап, на котором одним из обязательных условий является необходимость обеспечения быстрой работы с удаленными базами данных, в том числе через глобальную сеть.

Разработка приложения, управляющего работой баз данных, состоит из создания двух программных частей: серверной и клиентской.

Серверная часть приложения разрабатывается, как правило, средствами встроенного в соответствующие СУБД языка SQL (SQL-Server, Oracle, и др.).

Клиентская часть приложения разрабатывается, как правило, с использованием универсальных языков программирования.

Одним из средств разработки клиентской части приложения является объектно-ориентированный язык программирования Visual Basic.NET. Эта современная визуальная среда обеспечивает:

- простоту создания пользовательского интерфейса программы;
- возможность работы с Web-сервисами;
- создание клиент-серверных приложений (включая работу через Интернет);
- поддержку многоплатформенного протокола передачи данных — SOAP-протокола.

SOAP-протокол — это набор правил для работы с удаленными объектами. Где именно находятся эти удаленные объекты (в другом каталоге, в корпоративной интрасети или в сети Интернет) — для клиентских программ, использующих SOAP-протокол, абсолютно неважно. SOAP-протокол основывается на языке XML. Любая передаваемая информация между клиентом и сервером в этом случае является отдельным XML-документом, написанным по правилам SOAP-протокола.

SOAP-протокол — это слабосвязанный механизм, ориентированный на сообщения и предназначенный для удаленного вызова объектов по глобальным сетям, работа с удаленными базами данных в котором осуществляется с помощью HTTP-запросов и ответов.

Реализация СУБД — это этап, следующий после разработки «эскизного проекта» и приложения. На этапе реализации информационной системы фактически осуществляется формирование базы данных в конкретных условиях производства, т.е. происходят:

- формирование серверной части системы;
- формирование клиентской части системы;
- доработка программных модулей управления базой данных;
- установление прав доступа к таблицам баз данных и разработка других методов защиты информации;
- разработка инструкций, обучение пользователей и администраторов работе с базами данных.

Фактически данный этап жизненного цикла является началом сдачи разработанной системы заказчику.

Загрузка данных — это этап, который сводится к заполнению таблиц соответствующими данными в полном соответствии с разработанными на предыдущем этапе инструкциями пользователей. Если разработанная база данных должна функционировать совместно с другими информационными системами, необходимо обеспечить их взаимодействие, используя для этого и методы присоединения данных.

Тестирование — это этап, предназначенный для нахождения возможных ошибок при работе и управлении разработанной информационной системой. Очевидно, что для выполнения тести-

рования все таблицы базы данных должны быть заполнены соответствующей информацией.

Процесс тестирования можно осуществлять двумя способами:

- непосредственной проверкой функционирования системы пользователями и администраторами в соответствии с разработанными инструкциями;
- с помощью специальных экспертных программ, позволяющих автоматически находить ошибки, например при вводе данных в таблицы или запросы.

Эксплуатация и сопровождение — это этап, на котором предполагается непрерывное наблюдение за разработанной системой в процессе ее функционирования. Как правило, контроль качества работы системы осуществляет администратор базы данных. Очевидно, что процесс контроля качества системы должен полностью соответствовать действующим на предприятии методам системы менеджмента качеством, отвечающим требованиям стандартов ISO 1900:2000.

2.5. Администрирование баз данных

В практике разработки и управления базами данных выделяют две функции: администрирование данных и администрирование баз данных.

Соответственно предусматривают и две должности: администратор данных и администратор баз данных.

Администрирование данных подразумевает управление информационными ресурсами, разработку и внедрение стандартов на информационное обеспечение, концептуальное и логическое проектирование баз данных.

Приведем перечень задач, возлагаемых на администратора данных:

- выбор рациональных инструментальных средств разработки баз данных;
- участие в разработке корпоративных стратегий создания информационной системы с учетом развития информационных технологий, включая разработку соответствующей технической документации;
- экспертная оценка осуществимости проектов и планирование процесса создания базы данных;
- участие в разработке корпоративной модели данных;
- определение требований организации к используемым данным и разработка словаря данных;
- участие в разработке методик сбора данных и выбор формата их представления;
- оценка существующих и ожидаемых объемов данных;

• определение правил доступа к данным и мер безопасности, соответствующих правовым нормам и внутренним требованиям организации;

• концептуальное и логическое проектирование базы данных;

• взаимодействие с разработчиками приложений в целях обеспечения существующих требований конкретного предприятия;

• разработка должностных инструкций и обучение пользователей баз данных;

• взаимодействие со службами системы управления качеством в части анализа причин нарушения целостности БД или потери информации;

• разработка и выбор методов и средств эффективной защиты данных в условиях конкретного предприятия.

Администрирование баз данных подразумевает управление физической реализацией разработанной информационной системы.

Деятельность администратора баз данных является в большей мере технической, предусматривающей знание особенностей конкретных СУБД и операционных систем.

Приведем перечень задач администратора баз данных:

• участие в оценке и выборе целевой СУБД на этапах проектирования баз данных;

• физическое проектирование базы данных;

• реализация физического проекта базы данных в среде целевой СУБД;

• определение и реализация требований по защите и поддержке целостности данных;

• взаимодействие с разработчиками приложений баз данных;

• разработка стратегии тестирования баз данных;

• обучение пользователей при работе с базами данных в ЛВС предприятия;

• сдача в эксплуатацию готового приложения базы данных, контроль текущей производительности системы и соответствующая настройка базы данных;

• регулярное резервное копирование;

• разработка требуемых механизмов и процедур восстановления информации в базах данных;

• обеспечение полноты необходимой документации для работы с базами данных, получаемых как от внешних разработчиков, так и от специалистов внутри организации;

• поддержка и обеспечение работоспособности используемого программного и аппаратного обеспечения.

Контрольные вопросы

1. Дайте определение основного современного направления совершенствования производства и бизнеса — CALS-технологии.

2. Что означают следующие принципы разработки многопользовательских систем управления базами данных: учет интересов всех потенциальных пользователей систем и модульный принцип разработки и внедрения системы?

3. Назовите основные этапы проектирования многопользовательских баз данных.

4. Какие задачи необходимо решить при разработке проекта базы данных?

5. Назовите основные компоненты систем управления реляционными базами данных.

6. Назовите основные характеристики, достоинства и недостатки следующих форм организации многопользовательских баз данных: файл—сервер и клиент—сервер.

7. Какие задачи решает администратор данных?

8. Какие задачи должен решать администратор баз данных?

ЧАСТЬ II

СИСТЕМЫ РАЗРАБОТКИ И УПРАВЛЕНИЯ УДАЛЕННЫМИ БАЗАМИ ДАННЫХ

ГЛАВА 3

ТЕХНОЛОГИИ РАЗРАБОТКИ И УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ СРЕДСТВАМИ ЯЗЫКА SQL

3.1. Назначение языка SQL

Любой язык разработки и определенные базами данных должен предоставлять пользователю определенные возможности. Перечислим их:

- создание баз данных и таблиц с полным описанием их структуры;
- выполнение основных операций манипулирования данными, таких как вставка, модификация и удаление данных из таблиц;
- выполнение простых и сложных запросов.

При этом язык работы с базами данных должен решать все указанные задачи при минимальных трудовых и материальных затратах.

Кроме того, язык разработки и управления базами данных должен отвечать некоторому заданному стандарту, что позволит использовать один и тот же синтаксис и одинаковую структуру команд при переходе от одной СУБД к другой.

Язык SQL отвечает практически всем этим требованиям.

SQL является примером *языка преобразования данных*, или же языка, предназначенного для работы с таблицами в целях преобразования входных данных к требуемому выходному виду. Язык SQL, определенный стандартом ISO, включает в себя два основных компонента:

- язык DDL (Data Definition Language), предназначенный для определения структур базы данных и управления доступом к данным;
- язык DML (DataMunipulation Language), предназначенный для выборки и обновления данных.

Язык SQL — это специальный и пока единственный стандартный язык разработки и управления (манипулирования) реляционными базами данных, составляющий основу всех современных СУБД: Fox Pro, Microsoft Access, Oracle, SQL-Server и др.

3.2. Основные правила записи операторов

Стандартом ISO 1900:2000 в SQL установлены следующие термины, определяющие структуру базы данных: *таблица*, *столбец* и *строка*.

При написании кода программы рекомендуется использовать следующие правила записи операторов:

- каждая конструкция в операторе должна начинаться с новой строки;
- начало каждой конструкции оператора должно обозначаться одним и тем же отступом;
- если конструкция состоит из нескольких частей, каждая из них должна начинаться с новой строки с некоторым отступом относительно начала этой конструкции, что будет указывать на их подчиненность;
- для записи зарезервированных слов должны использоваться прописные буквы;
- для записи слов, определяемых пользователем, должны использоваться строчные буквы;
- вертикальная черта (|) указывает на необходимость выбора *одного* из *нескольких* приведенных значений, например *a* | *b* | *c*;
- фигурные скобки определяют *обязательный элемент*, например {*a*};
- квадратные скобки определяют *необязательный элемент*, например [*a*];
- многоточие (...) используется для указания многократного (необязательного) повторения выполнения группы операторов.

Например, запись {*a* | *b*} [, *c* ...] означает, что после *a* или *b* могут несколько раз повторяться символы *c*, разделенные запятыми.

На практике для определения структуры базы данных (в основном ее таблиц) применяются операторы языка DDL, а для заполнения таблиц данными и выбора из них информации с помощью запросов — операторы языка DML.

3.3. Операторы манипулирования данными

Для манипулирования данными применяются следующие операторы DML языка SQL:

SELECT — выборка данных из базы;
INSERT — вставка данных в таблицу;
UPDATE — обновление данных в таблице;
DELETE — удаление данных из таблицы.

SELECT — это один из основных операторов при выборе информации из таблиц баз данных.

В общем виде синтаксис инструкции SELECT можно описать следующим образом:

```
SELECT [ALL/DISTINCT] (Список полей таблицы или запроса)
    FROM (Список таблиц или запросов, на основе которых формируется запрос)
[WHERE (Условия отбора данных)]
  [GROUP BY (Список полей, выводимых в результате выполнения запроса]
[HAVING (Условия для группирования данных в запросе]
[ORDER BY (Список полей, по которым упорядочивается вывод данных в запросе]
```

В рассмотренной структуре инструкции SELECT:

ALL — ключевое слово, которое означает, что в результирующий набор записей включаются все записи таблицы или запроса, которые удовлетворяют условиям запроса;

DISTINCT — ключевое слово, которое означает, что в результирующий набор записей не включаются повторяющиеся записи таблицы или запроса.

Ключевые слова могут отсутствовать в запросе.

В табл. 3.1...3.6 представлены соответствующие группы операторов языка SQL и выполняемые ими действия.

Таблица 3.1

Операторы определения данных (DDL)

Оператор	Действие
CREATE TABLE	Создает новую таблицу БД
DROP TABLE	Удаляет таблицу из БД
ALTER TABLE	Изменяет структуру существующей таблицы или ограничения целостности, задаваемые для данной таблицы
CREATE VIEW	Создает виртуальную таблицу, соответствующую некоторому SQL-запросу
ALTER VIEW	Изменяет ранее созданное представление
DROP VIEW	Удаляет ранее созданное представление
CREATE INDEX	Создает индекс для некоторой таблицы в целях обеспечения быстрого доступа к ней по атрибутам, входящим в индекс
DROP INDEX	Удаляет ранее созданный индекс

Таблица 3.2

Операторы манипулирования данными (DML)

Оператор	Действие
DELETE	Удаляет одну или несколько строк, соответствующих условиям фильтрации, из базовой таблицы. Применение данного оператора согласуется с принципами поддержки целостности, поэтому он не всегда может быть выполнен корректно, даже если синтаксически записан правильно
INSERT	Вставляет одну строку в базовую таблицу. Допустимы модификации данного оператора, при которых сразу несколько строк могут быть перенесены из одной таблицы или запроса в базовую таблицу
UPDATE	Обновляет значения одного или нескольких столбцов в одной или нескольких строках, соответствующих условиям фильтрации

Таблица 3.3

Оператор запросов (DML)

Оператор	Действие
SELECT	Заменяет все операторы реляционной алгебры и позволяет сформировать результирующее отношение, соответствующее запросу

Таблица 3.4

Операторы управления действиями — транзакциями (DML)

Оператор	Действие
CCOMMIT	Завершает комплексную взаимосвязанную обработку информации, объединенную в транзакцию
ROLLBACK	Отменяет изменения, проведенные в ходе выполнения транзакции
SAVEPOINT	Сохраняет промежуточное состояние БД, т.е. помечает его для того, чтобы в дальнейшем можно было к нему вернуться

Таблица 3.5

Операторы администрирования данными (DDL)

Оператор	Действие
ALTER DATABASE	Изменяет набор основных объектов в базе данных и ограничений, касающихся всей базы данных
ALTER DBAREA	Изменяет ранее созданную область хранения
ALTER PASSWORD	Изменяет пароль для всей базы данных
CREATE DATABASE	Создает новую базу данных
CREATE DBAREA	Создает новую область хранения базы данных
DROP DATABASE	Удаляет базу данных
DROP DBAREA	Удаляет область хранения базы данных
GRANT	Предоставляет права доступа к базе данных или отдельным ее элементам
REVOKE	Лишает права доступа к базе данных или отдельным ее элементам

Таблица 3.6

Операторы управления (DDL)

Оператор	Действие
DECLARE	Определяет курсор для запроса, задает имя и определяет связанный с ним запрос к БД
OPEN	Открывает курсор и объект базы данных
FETH	Устанавливает курсор на определенную запись и считывает ее
CLOSE	Закрывает курсор и объект базы данных
PREPARE	Генерирует план выполнения запроса в соответствии с инструкцией SELECT
EXECUTE	Выполняет сгенерированный ранее запрос

Контрольные вопросы

1. Какие возможности должен предоставлять пользователю любой язык разработки и управления базами данных?
2. Из каких двух основных компонентов, определенных стандартом ISO, состоит язык SQL?
3. Назовите основные правила записи операторов языка SQL.

4. Назовите назначение следующих операторов: SELECT, INSERT, UPDATE, DELETE.

5. Поясните назначение операторов в следующей структуре:

```
SELECT [ALL/DISTINCT]
   FROM [WHERE]
   [GROUP BY]
[HAVING]
[ORDER BY]
```

6. Какие операторы определения данных, манипулирования данными, управления действиями (транзакциями) и администрирования данными вы знаете?

УПРАВЛЕНИЕ УДАЛЕННЫМИ БАЗАМИ ДАННЫХ В СИСТЕМЕ SQL SERVER2000

4.1. Службы управления базами данных SQL Server2000

SQL Server2000 — распространенная система управления удаленными базами данных. Рассмотрим основные службы SQL Server2000, их назначение, методы использования и системные базы данных.

Как и многие другие серверные продукты, работающие под управлением операционных систем Windows NT и Windows2000, Microsoft SQL Server2000 реализована в виде *набора служб операционной системы*, каждая из которых запускается самостоятельно и отвечает за определенный круг задач.

SQL Server2000 включает в себя следующие службы управления базами данных :

- MSSQLServer;
- SQLServerAgent;
- Microsoft Search (MSSearch);
- Microsoft Distributed Transaction Coordinator (MSDTC).

Организация СУБД в виде отдельных служб позволяет SQL Server2000 работать в виде части операционной системы, в том числе иметь собственные права доступа и не зависеть от пользователя, работающего на компьютере в данный момент времени.

Служба MSSQLServer. Данная служба является ядром системы SQL Server2000. В частности, в задачи этой службы входят:

- регистрация пользователей;
- контроль их прав доступа;
- установление соединения;
- обслуживание обращений пользователей к базам данных;
- выполнение хранимых процедур;
- работа с файлами баз данных;
- ведение журнала транзакций;
- контроль за использованием системных ресурсов.

При использовании многопроцессорных операционных систем для повышения производительности служба MSSQLServer выполняет распараллеливание запросов пользователей между всеми доступными процессорами.

Все остальные службы управления можно рассматривать как расширения службы MSSQLServer, добавляющие гибкость и функциональность СУБД SQL Server2000.

Служба MSSQLServer всегда запускается первой и только после ее загрузки могут быть запущены для работы другие службы.

Служба SQLServerAgent. Данная служба предназначена для автоматизации администрирования работы с удаленными базами данных. С ее помощью можно автоматически выполнять запуск различных задач в определенное время, что избавляет администратора от большей части рутинной работы.

Например, администратор может спланировать автоматическое выполнение операций резервного копирования и проверки целостности информации в базе данных во время наименьшей активности пользователей. При этом администратору не требуется находиться рядом с сервером и контролировать ход выполнения операций.

Большая часть операций, которые выполняются службой SQLServerAgent, реализована в виде хранимых процедур, выполняемых службой MSSQLServer.

В задачи службы SQLServerAgent входят:

- автоматический запуск заданий;
- извещение операторов о сбоях в работе сервера.

В работе службы SQLServerAgent применяются объекты трех типов:

- Jobs (задания);
- Operators (операторы);
- Alerts (события).

Объекты Jobs описывают задания, которые должны выполняться автоматически.

Для каждого задания указывают одно или несколько расписаний (schedule) его запуска, однако оно также может быть выполнено администратором и вручную по требованию (on demand).

Задание состоит из одного или нескольких шагов (step), в качестве которых могут выступать:

- команда или запрос Transact-SQL;
- команды управления подсистемой репликации;
- утилиты командной строки или приложения Windows;
- выполнение скрипта VBScript или JavaScript и др.

Служба SQLServerAgent позволяет создавать многошаговые задания, причем шаги могут быть связаны между собой по определенным правилам.

Например, можно разработать задание на проверку целостности базы данных, и если оно завершается успешно, то служба SQLServerAgent может создать резервную копию данных или же сервер может отправить соответствующее извещение администратору по электронной почте либо на пейджер.

Служба SQLServerAgent позволяет гибко управлять временем запуска задач, обеспечивая их выполнение как в определенное время, так и в моменты наименьшей загруженности сервера.

Объекты Operators описывают операторов — служащих, отвечающих за поддержание сервера в рабочем состоянии.

В небольших организациях функции оператора и администратора обычно совмещает один человек, а на больших предприятиях и в корпорациях эти функции обычно выполняют разные люди. Администратор выполняет только часть работы, связанную с управлением, например занимается планированием, созданием и изменением баз данных. Оператор чаще занимается рутинной работой: выполняет резервное копирование базы данных, добавляет новых пользователей, контролирует целостность данных и т.д.

Если организация большая, возможно использование специализированных операторов. Например, один оператор отвечает за выполнение операций резервного копирования, другой — следит за целостностью данных и т.д. Соответственно каждый из них должен получать сообщения, относящиеся к его виду деятельности, т.е. нежелательно, чтобы оператор резервного копирования разрешал проблемы мертвых блокировок.

Объекты Alerts описывают события, на которые должна реагировать система SQL Server2000. При наступлении описанного с их помощью события сервер посредством службы SQLServerAgent отправляет одному или нескольким операторам извещение об обнаружении неполадок в работе.

События системы SQL Server2000 охватывают почти все аспекты работы сервера, что позволяет эффективно контролировать ее работу.

Информация обо всех объектах SQL Server2000, включая расписание автоматического запуска задач, хранится в системной базе данных Msdb, содержание которой при каждом старте анализирует служба SQLServerAgent, и если к моменту запуска накопились просроченные задания или произошло сконфигурированное событие, она выполняет соответствующие действия.

Для управления заданиями, операторами и событиями можно использовать следующие методы:

- применение графического интерфейса утилиты Enterprise Manager, являющееся достаточно удобным и наглядным;
- вызов системных хранимых процедур и команд Transact-SQL;
- обращение к интерфейсу SQL-DMO, обеспечивающее возможность написания собственных приложений, для работы с заданиями, операторами и событиями.

Использование службы SQLServerAgent позволяет значительно снизить расходы на управление удаленными базами данных, в том числе за счет уменьшения числа операторов и администраторов.

Служба MSS. Данная служба, также называемая Full-Text Search, используется для поиска символьной информации в таблицах баз данных системы SQL Server2000.

Служба Microsoft Search обеспечивает выполнение *полнотекстового поиска* (full-text search), технология которого позволяет находить не только слова и фразы, идентичные указанным, но и близкие к ним по смыслу и написанию. После выполнения такого поиска пользователь получает также различные формы глаголов и существительных.

Для реализации полнотекстового поиска в SQL Server2000 существуют *полнотекстовые каталоги* (full-text catalog) и *полнотекстовые индексы* (full-text index), данные которых хранятся отдельно от основных данных в специальных файлах. Все действия по работе с этими файлами осуществляет служба MSSearch. Связь между службами MSSQLServer и MSSearch производится через специального поставщика (full-text provider).

Служба MSSearch периодически анализирует содержание таблиц баз данных и обновляет (repopulation) полнотекстовые каталоги и индексы.

Если необходимо создать полнотекстовый индекс заново, выполняют его перестроение (rebuild), из чего следует, что данными полнотекстового поиска надо управлять отдельно от основных данных. При этом администратор должен настроить интервалы обновления данных полнотекстового поиска. Кроме того, операции резервного копирования и восстановления файлов полнотекстового поиска необходимо выполнять отдельно от основных данных.

Служба MSDTC. Система SQL Server2000 дает возможность пользователям работать одновременно с несколькими источниками данных. Пользователи в одном запросе могут обращаться к различным базам данных, хранящимся на одном или разных серверах. Кроме того, пользователи могут обращаться не только к серверам Microsoft SQL Server2000, но и к любым источникам данных, работающим с технологией OLE DB. Эта технология позволяет использовать не только реляционные источники данных, такие как Oracle, FoxPro, MS Access, но и нереляционные источники данных: текстовые файлы, книги MS Excel и настольные приложения.

Для обращения из тела одной транзакции к множеству источников данных система SQL Server2000 использует *распределенные транзакции* (distributed transaction), управление которыми осуществляет координатор распределенных транзакций (Distributed Transaction Coordinator).

В SQL Server2000 координатор распределенных транзакций реализован в виде службы MSDTC. Эта служба автоматически отслеживает ситуации, в которых необходимо начать выполнение распределенных транзакций. При этом в некоторых ситуациях пользователь может и не подозревать, что его транзакция выполняется как распределенная, так как служба MSDTC скрывает от

пользователя все действия по обработке распределенных транзакций.

Распределенные транзакции реализуются как множество локальных транзакций, открываемых в каждом источнике данных координатором распределенных транзакций. При этом служба MSDTC, используя специальный *двухфазный протокол изменений* (2PC, two-phase commit protocol), синхронизирует все транзакции таким образом, что пользователь может быть уверен в целостности данных во всех участниках распределенной транзакции.

4.2. Системные базы данных SQL Server2000

SQL Server2000 в своей работе использует несколько системных баз данных, которые создаются автоматически при ее установке и не должны удаляться. Вся информация о настройке сервера хранится в этих базах данных. Их можно сравнить с реестром операционной системы Windows, в котором хранится вся системная и пользовательская информация и удаление или повреждение которого приведет к разрушению системы. Аналогичная ситуация наблюдается и со следующими системными базами данных SQL Server2000: Master, Model, Tempdb и Msdb.

База данных Master. Является главной базой данных SQL Server2000, выполняющей функции реестра операционной системы Windows. Остальные системные базы данных имеют второстепенное значение и их можно считать вспомогательными. В базе данных Master хранится вся системная информация о параметрах конфигурации сервера, имеющихся на сервере пользовательских базах данных, пользователях, имеющих доступ к серверу, и др.

По умолчанию база данных Master создастся в каталоге Data установочного каталога системы SQL Server2000. Состоит данная база данных из двух следующих файлов:

- Master.mdf — основной файл, содержащий собственно данные, размер которого после установки составляет 8 Мбайт;
- Master.ldf — файл, предназначенный для хранения журнала тразакций, размер которого после установки составляет 1 Мбайт.

База данных Model. Является шаблоном для создания новых баз данных. Технология создания новой базы данных в SQL Server2000 следующая: сервер копирует базу данных Model в указанное место и изменяет ее имя соответствующим образом. Если при создании базы данных не указаны никакие другие параметры кроме ее имени, то новая база данных будет являться полной копией базы данных Model. Если же указаны размер и состав файлов создаваемой базы данных, то соответственно изменится скопированная база данных. Но в любом случае в качестве основы используется база данных Model.

Изменяя параметры базы данных Model, можно управлять параметрами по умолчанию создаваемых баз данных. Кроме того, базу данных Model можно использовать в качестве корпоративного стандарта на содержимое и свойства создаваемой базы данных. Администратор может создать в базе данных Model набор таблиц и хранимых процедур, которые должны быть в каждой базе данных, а не утруждать себя изменением очередной вновь созданной базы данных вручную.

Можно ускорить создание множества однотипных таблиц со специализированной конфигурации, если соответствующим образом изменить базу данных Model.

После установки SQL Server2000 размер базы данных Model составляет 1,5 Мбайт. Располагается база данных Model в каталоге Data и состоит из следующих двух файлов размером по 0,75 Мбайт каждый:

- Model.mdf — основной файл, содержащий собственно данные;
- Model.ldf — файл, используемый для хранения журнала транзакций.

База данных Tempdb. Полное название этой базы данных Temporary DataBase. Она служит в SQL Server2000 для хранения всех временных объектов, создаваемых пользователями во время сеанса работы.

Пользователям иногда необходимо создавать временные таблицы, представления, курсоры и другие объекты для сохранения промежуточных результатов, что и позволяет делать SQL Server2000. Например, для создания временной таблицы достаточно добавить перед ее именем символ # или ##, и сервер автоматически создаст временную таблицу.

Временные объекты могут быть локальными и глобальными.

Локальные объекты доступны только из того соединения, в котором они созданы, но при этом можно создавать одноименные объекты в различных соединениях. Для организации локальных временной таблицы и представления в имя объекта следует добавить символ #, а для создания локальной переменной предназначен символ @.

Глобальные объекты, созданные в одном соединении, доступны из всех остальных активных соединений. При этом допускается создание единственного глобального временного объекта с уникальным именем. Для создания глобальных временной таблицы и представления в имя объекта следует добавить символ ##, а для создания локальной переменной используется символ @@.

Если постоянные объекты, такие как таблицы или представления, создаются в пользовательской базе данных, то временные объекты возникают в базе данных Tempdb. Доступ к базе данных Tempdb автоматически имеется у всех пользователей, т. е. admini-

стратор не должен предпринимать никаких действий для предоставления им доступа к этой базе данных.

Отличительной особенностью базы данных Tempdb является то, что она уничтожается каждый раз, когда происходит останов сервера. Естественно, все временные объекты, созданные пользователями, также уничтожаются. При следующем запуске SQL Server2000 база данных Tempdb создается заново. Понятно, что создание резервной копии базы данных Tempdb совершенно бесполезно.

При создании базы данных Tempdb так же, как и для пользовательских баз данных, в качестве основы применяется база данных Model. При этом наследуются и все свойства последней. Администратор должен учитывать это, изменяя параметры базы данных Model. Неверное конфигурирование параметров этой базы данных может оказать неблагоприятное влияние на работу всех пользователей. Кроме того, при планировании параметров базы данных Tempdb следует учитывать требования к свободному пространству на диске. Как и для всех баз данных, для Tempdb поддерживается возможность автоматического роста ее файлов, т.е. при интенсивном обращении пользователей к ресурсам базы данных Tempdb неизбежен ее рост. Однако необходимо правильно выбрать первоначальный размер и шаг прироста этой базы данных, так как неверное конфигурирование этих параметров может заметно снизить производительность системы.

База данных Tempdb включает в себя два следующих файла, располагающихся в каталоге Data установочного каталога SQL Server2000:

- Tempdb.mdf — основной файл, содержащий временные объекты, размер которого после установки составляет 8 Мбайт;
- Tempdb.ldf — файл, в котором хранится журнал транзакций и размер которого после установки составляет 0,5 Мбайт.

База данных Msdb. Предназначена для хранения всей информации, относящейся к автоматизации администрирования и управления системы SQL Server2000, а также информации об операторах и событиях. Кроме того, в этой базе данных хранится информация о расписании автоматического запуска заданий, т.е. в базе данных Msdb размещается вся системная информация, используемая службой SQLServerAgent.

4.3. Инструменты администрирования серверами SQL Server2000

Обычно инструменты администрирования устанавливаются при инсталляции SQL Server2000, однако они могут быть добавлены и отдельно. При этом возможно, что на одном компьютере будут

находиться только инструменты администрирования, а на другом — только собственно система SQL Server2000.

Инструменты системы SQL Server2000 спроектированы таким образом, что они могут применяться для работы с любым ее сервером в локальной сети предприятия, в том числе и с серверами SQL Server 7.0 или SQL Server 6.x. Однако администрирование этих серверов должно выполняться инструментами администрирования, поставляемыми в составе этих версий.

Рассмотрим следующие инструменты администрирования SQL Server2000: Enterprise Manager, SQL Server Service Manager, SQL Server Profiler, Query Analyzer, Upgrade Wizard, Import and Export Data.

Enterprise Manager. Данный инструмент является базовым при выполнении следующих задач:

- управление системой безопасности;
- создание баз данных и ее объектов;
- создание и восстановление резервных копий;
- конфигурирование подсистемы репликации;
- управление параметрами работы служб SQL Server2000;
- управление подсистемой автоматизации;
- запуск, останов и приостанов служб;
- конфигурирование связанных и удаленных серверов;
- создание, управление и выполнение пакетов DTS.

Приведенный список не исчерпывает всех областей применения Enterprise Manager и легко может быть расширен. Однако и указанных пунктов достаточно, чтобы понять всю важность этого инструмента.

Большая часть административных задач SQL Server2000 может быть выполнена следующими методами:

- использованием средств Transact-SQL;
- с помощью графического интерфейса Enterprise Manager;
- с помощью мастеров (wizards).

Порядок перечисленных методов соответствует уменьшению сложности работы с ними, т.е. самым сложным является выполнение задачи средствами Transact-SQL, так как это требует знания синтаксиса команд и хранимых процедур, а также умения обращаться с инструментом Query Analyzer (или любым другим ему подобным). Однако использование средств Transact-SQL открывает пользователю прямой доступ к системным данным.

Инструмент Enterprise Manager разрабатывался с целью облегчить пользователям выполнение наиболее актуальных административных задач за счет сочетания простоты работы с ним и его высокой функциональности. Можно с уверенностью сказать, что разработчикам Microsoft удалось добиться хорошего результата. Enterprise Manager является действительно достаточно простым в использовании инструментом, который, однако, охватывает прак-

тически все административные задачи, с которыми столкнется администратор. Конечно, различные нестандартные ситуации с помощью этого инструмента не решить — для этого придется обратиться к средствам Transact-SQL. Не стоит воспринимать Enterprise Manager и в качестве инструмента для неопытных пользователей, не умеющих работать со средствами Transact-SQL, и всеми силами стремиться осилить синтаксис команд и хранимых процедур.

Решить некоторые задачи средствами Transact-SQL настолько сложно, что это просто становится неразумным. Инструмент Enterprise Manager поможет сэкономить опытному пользователю много времени и использовать его более эффективно.

SQL Server Service Manager. Единственной задачей данного инструмента является предоставление пользователю удобного механизма запуска и останова служб SQL Server2000. Кроме того, он позволяет запретить или разрешить автоматический запуск той или иной службы при загрузке операционной системы.

Утилита Service Manager устанавливается при инсталляции системы SQL Server2000 и по умолчанию автоматически запускается при загрузке операционной системы. В нормальном состоянии утилита Service Manager представлена значком в правой части панели задач (taskbar). Двойным щелчком мышью на ее пиктограмме открывается окно программы, с помощью которого можно запускать, останавливать и приостанавливать службы SQL Server2000, а также разрешать или запрещать их автоматический запуск при загрузке операционной системы.

SQL Server Profiler. Это графический инструмент, с помощью которого администратор может наблюдать за теми или иными аспектами работы SQL Server2000. В основе работы этой утилиты лежит тот же принцип, что и в основе работы утилиты Performance. При выполнении пользовательских запросов, хранимых процедур, команд Transact-SQL, подключении к серверу и отключении от него, а также при множестве других действий ядро SQL Server2000 сохраняет в системных таблицах массу различной информации о ходе выполнения операций. Эту информацию можно получить с помощью специальных хранимых процедур. Утилита SQL Server Profiler использует эти хранимые процедуры для получения необходимой информации. Полученные данные затем предоставляются в удобном виде с помощью графического интерфейса. Однако пользователи могут получать информацию о процессах SQL Server2000, обращаясь напрямую к хранимым процедурам. В принципе, на основе этих хранимых процедур можно даже написать свое собственное приложение, которое будет отображать информацию о работе SQL Server2000 в требуемой форме.

Мониторинг работы SQL Server2000 основывается на наблюдении за событиями (events). Событие, генерируемое ядром SQL

Server2000, соответствует минимальному объему работы, который можно контролировать. Каждое событие принадлежит к какому-то классу событий (event classes), который описывает его параметры и смысл той или иной информации. Для лучшего понимания разницы между событиями и классами событий SQL Server Profiler проведем аналогию с объектами и экземплярами объектов Performance Monitor. Класс событий SQL Server Profiler, как и объект Performance Monitor, представляет собой абстрактное описание, а само событие (экземпляр объекта) — информацию о работе того или иного объекта.

Число классов событий SQL Server довольно велико, и для облегчения работы они были разделены на 12 категорий (category).

Query Analyzer. Этот инструмент предназначен для выполнения запросов и анализа их исполнения. По частоте использования и важности Query Analyzer сравним с Enterprise Manager. Кроме того, он обеспечивает проведение трассировки выполнения хранимых процедур.

При выполнении трассировки можно использовать точки останова (break points), а также осуществлять пошаговое выполнение команд процедуры.

Помимо выполнения запросов и хранимых процедур с помощью Query Analyzer можно оценивать производительность исполнения запроса, для чего следует разрешить отображение оценочного или результирующего плана исполнения запроса. Отметим, что *оценочный план* (estimated plan) исполнения запроса формируется на основе предположений сервера о затратах на выполнение отдельных шагов этого запроса. *Результирующий план* (execution plan) исполнения запроса генерируется после выполнения данного запроса и отражает реальное положение дел. Конечно, в идеальной ситуации значения оценочного и результирующего планов исполнения должны совпадать. Однако при работе с многопользовательскими системами вполне может оказаться, что реально исполнение запроса займет больше времени, чем ожидалось. Чаще всего это происходит из-за занятости процессора выполнением запросов других пользователей или блокированием необходимых для выполнения запроса ресурсов другими транзакциями.

Upgrade Wizard. Данный мастер предназначен для выполнения обновления баз данных от SQL Server 6.5 до SQL Server2000. В процессе обновления на SQL Server2000 переносятся собственно данные, а также весь набор объектов обновляемой базы данных, включая хранимые процедуры, триггеры, правила, умолчания, ограничения целостности, представления. Кроме того, также окажутся перенесенными пользователи базы данных со всеми установленными правами доступа к ее объектам и т. д. В процессе обновления также копируются все настройки подсистемы репликации.

Import and Export Data. Этот инструмент является мастером импорта (экспорта) данных, предназначенным для создания пакета DTS, который будет выполнять копирование информации между двумя источниками данных. Отличительной особенностью данного мастера является простота конфигурирования процесса копирования данных. К недостаткам его использования относится невозможность обработки более двух источников данных, а также невозможность определения сложных преобразований и отношений предшествования. Кроме того, большая часть возможностей DTS, например отправка сообщений по электронной почте, будет недоступна. Тем не менее несомненным достоинством использования данного мастера является легкость решения простых задач, т.е. для внесения в таблицу базы данных информации из файла MS Excel его возможностей будет вполне достаточно. Таким образом, даже неопытные пользователи могут выполнять основные операции обмена данными.

Утилиты конфигурирования сетевых параметров компьютерного пользователя и сервера Client Network Utility и Server Network Utility. Для того чтобы клиенты могли установить сетевое соединение с сервером, в клиентской и серверной частях необходимо добавить специальные сетевые библиотеки (Network Library). Эти библиотеки реализуются в виде динамически подключаемых библиотек (DLL — Dynamic Link Library) и подключаются к операционной системе. Библиотека расширяет базовые возможности сетевого протокола и является как бы надстройкой над ним, выполняющей различные сетевые операции по обмену данными между клиентом и сервером, для чего используются механизмы IPC.

Библиотеки можно инсталлировать как в процессе установки SQL Server2000, так и позже. Если требуется добавить или удалить библиотеку уже после установки системы, следует использовать утилиту Server Network Utility, устанавливаемую вместе с SQL Server2000. С помощью этой библиотеки конфигурируются сетевые параметры собственно сервера, т.е. указываются сетевые библиотеки, с помощью которых пользователи смогут обращаться к серверу. Однако со стороны клиента также требуется присутствие сетевых библиотек и конфигурирование их для работы с сервером. Конфигурирование библиотек клиента выполняется с помощью утилиты Client Network Utility, добавляемой при установке инструментов администрирования SQL Server. Сконфигурированные параметры используются для работы Enterprise Manager, Query Analyzer и других инструментов администрирования. Чтобы гарантировать успешное взаимодействие клиента с сервером, следует обеспечить использование клиентом хотя бы одной библиотеки, поддержка которой разрешена на сервере, а также при необходимости соответствующим образом указать ее свойства.

Утилиты командной строки. Помимо рассмотренных утилит, имеющих графический интерфейс, в SQL Server2000 существует набор утилит командной строки, с помощью которого также можно решать различные задачи. Некоторые из этих утилит используются сервером автоматически и скорее являются частью ядра SQL Server2000, чем собственно утилитами.

Контрольные вопросы

1. Каково основное назначение следующих служб SQL Server: MSSQLServer, SQLServerAgent, Microsoft Search и Microsoft Distributed Transaction Coordinator?
2. Каково основное назначение следующих системных баз данных SQL Server: Master, Model, Tempdb и Msdb?
3. Какие инструменты SQL Server2000 вы знаете?

ГЛАВА 5

УПРАВЛЕНИЕ УДАЛЕННЫМИ БАЗАМИ ДАННЫХ В СИСТЕМЕ ORACLE

5.1. Основные понятия и термины

Рассмотрим основные понятия и термины, используемые в системе Oracle — системе управления распределенными базами данных, разработанной фирмой Oracle Corporation, которые несколько отличаются от терминологии СУБД SQL Server и Microsoft Access.

Триггер — механизм, позволяющий создавать процедуры, которые будут автоматически запускаться при выполнении команд INSERT, UPDATE или DELETE.

Транзакция — логически завершенный фрагмент последовательности действий (одна или более SQL-команд, завершенных фиксацией или откатом).

Объекты схемы — абстракции, составляющие базы данных. Это индексы, кластеры, пакеты, последовательности, хранимые процедуры, синонимы, таблицы, представления и т.д.

Таблица — основная единица хранения данных БД Oracle. Включает в себя имя, строки и столбцы. Каждый столбец также имеет имя и тип данных. Таблицы хранятся в табличных пространствах, причем часто в одном табличном пространстве находятся несколько таблиц.

Кластер — набор таблиц, физически хранящихся как одна и имеющих общие столбцы. Использование кластеров крайне эффективно, если часто обрабатываются запросы к данным двух и более таблиц, имеющих общие столбцы. К таким таблицам можно обращаться по отдельности даже в том случае, если они являются частью кластерной таблицы.

Индекс — структура, позволяющая извлекать данные быстро и эффективно (точно так же, как оглавление какой-либо книги позволяет найти интересующий раздел). Индекс объявляется для одного или нескольких столбцов. Доступ к таблице происходит по проиндексированному столбцу (столбцам).

Представление (*вид*) — окно (рамка) для просмотра данных из одной или более таблиц. Вид не хранит никаких данных, а только представляет их. С видами возможны те же операции, что и с таблицами (построение запросов, обновление, удаление) без всяких ограничений. Представления часто используются для упрощения восприятия пользователем хранящихся в базе данных посредством

извлечения из таблицы лишь части необходимых данных или набора данных из нескольких таблиц. Кроме того, представления могут использоваться для ограничения доступа пользователей к некоторым данным.

Хранимая процедура — SQL-запрос, хранимый в словаре данных. Хранимые процедуры разрабатываются для эффективного выполнения запросов. При использовании хранимых процедур можно уменьшить сетевой трафик СУРБД и тем самым увеличить ее производительность.

Буфер — некоторый объем оперативной памяти, используемый для хранения данных. Содержит данные, которые предполагается использовать или которые использовались совсем недавно. В большинстве случаев буфер содержит копию данных, которые хранятся на жестком диске. Данные в буфере могут изменяться, записываться на диск, а также временно храниться. В системе Oracle буферы содержат те блоки данных, к которым недавно обращались. В буфере журнала изменений сохраняются временные записи журнала изменений, которые затем записываются на диск.

Кэш буферов данных — совокупность буферов или область памяти для быстрого доступа к данным. С точки зрения аппаратного обеспечения — это небольшой (применительно к оперативной памяти) объем памяти, который значительно быстрее основной памяти и который используется для уменьшения времени, необходимого на частую загрузку данных или инструкций в центральный процессор (ЦП), имеющий встроенный кэш.

Последовательность — генератор последовательностей, используемый для создания последовательности цифр, хранимых в кэш буферов данных.

DBWR (DataBase WRiter) — процесс, основная задача которого записывать изменения базы данных на физический жесткий диск.

Чистый буфер (clean buffer) — буфер, содержимое которого не подвергалось изменению, а следовательно, нет необходимости записывать его на жесткий диск.

Грязный буфер (dirty buffer) — буфер, содержимое которого изменилось. (DBWR периодически сбрасывает грязные буферы на жесткий диск.)

SGA (System Global Area) — разделяемая область памяти, используемая для хранения данных и управляющей информации экземпляра Oracle. Размещается в памяти при запуске экземпляра Oracle и освобождается при завершении его работы. SGA составляют буферы данных, буфер журнала изменений и разделяемый пул (shared pool).

Блок (block) — самая маленькая единица хранения данных в СУРБД Oracle. Содержит информацию заголовка и сами данные или PL/SQL-код. Размер блока от 2 до 16 Кбайт.

Узкое место (bottleneck) — компоненты, ограничивающие производительность или эффективность системы.

Словарь данных (data dictionary) — набор таблиц, используемых для поддержания информации о БД.

Контрольная точка (checkpoint) — операция, приводящая к тому, что все измененные данные (блоки данных в памяти) записываются на диск. Является ключевой операцией при необходимости быстрого восстановления базы данных после сбоя.

Схема (schema) — коллекция объектов БД.

Конкурирование (concurrency) — способность программы выполнять несколько функций одновременно. Применительно к Oracle — это возможность одновременного доступа к данным для множества пользователей.

DDL (Data Definition Language) — язык описания данных. Команды этого языка предназначены для создания, изменения и удаления объектов базы данных. В системе Oracle команды DDL связаны с администрированием баз данных, т.е. перед и после выполнения каждой DDL-команды система обязательно фиксирует все текущие транзакции (чтобы избежать потерь информации).

DML (Data Manipulation Language) — язык манипулирования данными. Команды этого языка позволяют строить запросы и оперировать данными существующих объектов схемы. В отличие от DDL фиксирование транзакций после каждой команды в этом языке не производится. Существуют следующие команды DML: DELETE, INSERT, SELEC, UPDATE, EXPLAIN PLAN; LOCK TABLE.

Динамические таблицы характеристик (dynamic performance tables) — таблицы, которые автоматически создаются при запуске экземпляра Oracle и используются для хранения характеристик этого экземпляра. Они включают в себя информацию о соединениях, вводе-выводе, первоначальные значения параметров среды и др.

Процедура — набор SQL или PL/SQL-команд, выполняющих определенную задачу. Процедура может иметь входные параметры, но не имеет выходных.

Функция — совокупность SQL или PL/SQL-команд, реализующих определенную задачу. В отличие от процедуры функция возвращает какое-либо значение переменной. Создание функций позволяет уменьшить число инструкций, передаваемых по сети.

Программный блок — относительно СУРБД Oracle программа, используемая для описания пакета, хранимой процедуры или последовательности процедур.

Запрос — транзакция *Только для чтения*. Генерируется с помощью команды SELECT. В отличие от обычной транзакции при запросе данные не изменяются.

Приведем компоненты СУБД Oracle, характерные для различных ее модификаций.

OLTP (On-line Transaction Processing) — система оперативной обработки транзакций. Эти системы обеспечивают работу большого числа пользователей, работающих с многопользовательскими базами данных, т.е. быстрые ответы на запросы всех клиентов.

DSS (Decision Support System) — система поддержки принятия решений, которые используются в процессах интеллектуального анализа данных. Эти системы выполняют множество запросов, связанных с обработкой больших объемов информации, хранящейся в разных таблицах и разных базах данных.

Хранилище данных (Data Warehouse) — крупномасштабная система, хранящая результаты работы систем OLTP и DSS, т.е. хранящая и обрабатывающая информацию, занимающую многие сотни гигабайт памяти.

Информационная лавка (Data Mart) — уменьшенная версия хранилища данных (Data Warehouse), как правило, ориентированная на решение специализированных задач. Обеспечивает хранение и обработку информации, требующей менее сотни гигабайт памяти.

Видеосервер — сервер, предназначенный для обработки видеоинформации. Имеет широкую полосу пропускания для поддержания большого количества видеопотоков. Должен справляться с большой нагрузкой ввода-вывода, так как при считывании с устройств загружаются сразу большие блоки данных.

Веб-сервер — сервер, предназначенный для работы с статическими и динамическими веб-страницами, которые могут быть как очень простыми, так и комплексными, генерируемыми из баз данных. Веб-сервер Oracle, как правило, используется для коммерческих веб-приложений, позволяющих покупателям просматривать каталоги, содержащие изображения товаров и даже видеоиллюстрации. Обычно он поддерживает значительное количество пользователей и имеет большой объем данных. Его производительность зависит от объема оперативной памяти.

OLAP (On-line Analytical Processing) — система аналитической обработки информации в реальном масштабе времени. Как правило, ее пользователями являются финансовые аналитики или маркетинговый персонал, работающий с данными на глобальном уровне.

5.2. Типы пользователей

Типы пользователей Oracle и их обязанности зависят от конфигурации системы и конкретной организации ее корпоративной базы данных. Например, в крупных системах обязанности администратора базы данных могут распределяться среди нескольких специалистов. В то же время в небольших системах один человек

может одновременно выполнять функции нескольких типов пользователей.

Можно выделить следующие основные типы пользователей, характерные для всех систем управления базами данных:

- администратор базы данных;
- администратор по защите данных;
- разработчик приложения;
- администратор приложения;
- пользователь базы данных;
- администратор сети.

Администратор базы данных (DataBase Administrator — DBA) — специалист, управляющий работой базы данных. Обычно обязанности DBA подразделяют на основные и дополнительные.

Основные обязанности DBA следующие.

- *Установка нового программного обеспечения.* Установка новых версий Oracle, приложений и другого программного обеспечения, относящегося к администрированию СУБД. Предусматривает также обязательное тестирование устанавливаемых программ перед введением их в рабочую среду.
- *Конфигурирование программного и аппаратного обеспечения.* В большинстве случаев доступ к настройке программного и аппаратного обеспечения имеет только системный администратор, поэтому DBA должен производить установку программ, конфигурирование программного и аппаратного обеспечения только совместно с системным администратором.
- *Обеспечение безопасности.* Является одной из основных обязанностей DBA. Управление безопасностью и администрирование включают в себя: добавление и удаление пользователей, управление квотами, аудит и разрешение проблем безопасности.
- *Настройка производительности.* Даже хорошо настроенная система нуждается в постоянной проверке производительности и периодической ее перенастройке. Иногда для этого достаточно изменить параметры системы или индексы, а может быть, перестроить структуру таблиц.
- *Резервное копирование и восстановление системы.* Одна из главных обязанностей DBA — постоянно сохранять данные в системе. Чтобы делать это эффективно, необходимо разработать процедуру резервного копирования и стратегию восстановления данных. Очень важно периодически тестировать отработанную схему резервного копирования и восстановления данных.
- *Процедура постоянного (планового) обслуживания.* Обслуживание СУБД лучше всего производить рано утром либо по выходным дням, чтобы не нарушать работу пользователей. Обслуживание включает в себя: архивирование, тестирование и настройку системы. Администратор должен составить календарь планового обслуживания СУБД и довести его до сведения клиентов.

• *Локализация неисправностей и восстановление системы после сбоя.* Поскольку сбой системы приводит к возможности потери доступа пользователей к своим данным, DBA обязан как можно быстрее восстановить работу системы, т.е. он должен уметь предусмотреть сбой и заранее иметь план восстановления системы после сбоя.

Дополнительные обязанности DBA сводятся, как правило, к оказанию помощи отдельным клиентам и могут включать в себя следующие задачи администрирования.

• *Анализ данных.* Проводится для того, чтобы дать отдельным разработчикам или пользователям рекомендации по повышению производительности или эффективности хранения данных.

• *Предварительная разработка БД.* Поскольку DBA знает систему «изнутри», он может на предварительной стадии разработки структуры БД указать команде разработчиков на потенциальные проблемы и помочь в увеличении производительности программ.

• *Оказание консультаций разработчикам по хранимым SQL-процедурам.* DBA довольно часто привлекается к разрешению проблем SQL-кода и разработке (написанию) хранимых процедур, т.е. он должен быть готов стать консультантом для разработчиков и пользователей.

• *Разработка производственных стандартов и соглашений по именам.* Это одна из основных организационных проблем управления. Поскольку в разработке и развертывании приложений могут принимать участие несколько различных групп, DBA должен принимать активное участие в решении проблемы их соответствия производственным стандартам и соглашениям по именам.

• *Документирование среды.* DBA должен документировать каждый аспект среды СУБД, включая конфигурирование оборудования, обновление и изменение программного обеспечения, вопросы, связанные с изменением системы и ее параметров, и уметь полностью восстановить ее по документации в случае необходимости.

• *Планирование нагрузки системы и необходимого объема памяти.* Неотъемлемой частью работы DBA является определение необходимости приобретения дополнительных серверов, дополнительной дисковой и оперативной памяти в целях удовлетворения возрастающих потребностей пользователей. Прогнозируя ожидаемую потребность аппаратных средств, администратор обеспечивает надежность работы информационной системы предприятия.

5.3. Физическая архитектура хранения данных

СУРБД Oracle, предназначенная для одновременного доступа к большим объемам (терабайтам) хранимой информации, скла-

дывается из двух составляющих: базы данных (информации) и экземпляра (конкретной реализации системы).

База данных. Состоит из физических файлов, хранящихся в системе, и логических файлов (например, схемы БД). Физические файлы хранятся на диске, а логические файлы являются компонентами физического уровня.

Итак, базы данных Oracle состоят из двух уровней: физического и логического.

Физический уровень БД включает в себя три категории файлов: файлы данных, файлы журналов операций, управляющие файлы.

- *Файлы данных* хранят информацию, имеющуюся в БД. Причем в БД может храниться и один файл данных и сотни. Информация из одной таблицы может быть распределена по нескольким файлам данных, а также несколько таблиц могут делить между собой пространство одного файла данных. При этом распределение таблиц по нескольким файлам данных может значительно увеличить производительность системы. Количество файлов данных ограничивается параметром maxdatafiles.
- *Файлы журналов операций* (redo log files) содержат информацию, необходимую для процесса восстановления в случае сбоя системы, и все изменения, которые произошли в базе данных. С помощью журнала операций восстанавливают те изменения, которые были произведены, но не зафиксированы перед сбоем системы, поэтому файлы журналов операций должны быть очень хорошо защищены от аппаратных сбоев (как на программном, так и на аппаратном уровне). Если информация журнала операций будет утеряна, восстановить систему будет практически невозможно.
- *Управляющие файлы* содержат информацию, необходимую для запуска экземпляра Oracle (в том числе расположение файлов данных и файлов журналов операций), и они также должны быть хорошо защищены.

Логический уровень БД составляют следующие элементы: табличные пространства и схемы БД.

- *Табличные пространства* — это одна или несколько логических частей, на которые подразделяется база данных и которые используются для логической группировки данных между собой. Например, можно определить одно табличное пространство для бухгалтерских данных, а другое — для складских. Сегментирование групп по табличным пространствам упрощает администрирование этих групп. Каждое табличное пространство может состоять из одного или многих файлов данных. Используя несколько файлов данных для одного табличного пространства, можно распределить их по разным дискам, увеличив тем самым скорость ввода-вывода и соответственно производительность системы.

• *Схемы БД* — специальные логические структуры, с помощью которых в СУРБД Oracle происходит контроль над дисковым пространством. Эти структуры состоят из блоков данных, экстентов, сегментов.

Блок данных — это наименьшая единица хранения данных в БД Oracle. Блоки данных, содержащие заголовочную информацию о себе и данные, физически хранятся на диске и в большинстве систем занимают 2 Кбайт (2 048 байт), но для увеличения эффективности работы системы этот размер можно изменить.

Экстенты состоят из блоков данных и являются строительными блоками сегментов. Используются они для минимизации неиспользуемого (пустого) пространства хранилища. По мере увеличения количества данных в табличных пространствах экстенты используются для хранения тех данных, которые могут разрастаться. Таким образом, несколько табличных пространств могут делить между собой пространство хранилища без предварительного определения их разделов.

При создании табличного пространства можно указать минимальное число экстент, что позволит контролировать все пространство хранилища БД.

Сегменты, в свою очередь, состоят из совокупностей экстентов, содержащих определенный вид данных. БД Oracle использует четыре типа сегментов:

сегмент данных, хранящий пользовательские данные;

индексный сегмент, содержащий индексы;

сегмент отката, хранящий информацию отката, используемую при возврате к предыдущему состоянию БД;

временный (*промежуточный*) *сегмент*, создаваемый в случае если для выполнения SQL-выражения необходимо дополнительное рабочее пространство и уничтожаемый сразу после выполнения SQL-команд. Промежуточные сегменты используются также в различных операциях с БД, например при сортировке.

Экземпляр. Представляет собой конкретный способ доступа к данным и состоит из разделяемой памяти и процессов.

Разделяемая память (shared memory) используется для кэширования данных и индексов, а также для хранения программного кода. Разделяемая память подразделяется на несколько частей (или структур памяти), основными из которых являются: системная глобальная область (System Global Area) и программная глобальная область (Program Global Area).

• *Системная глобальная область* (*SGA*) — это область разделяемой памяти, которая используется для хранения данных и управляющей информации одного конкретного экземпляра Oracle. SGA размещается в памяти при запуске экземпляра Oracle и освобождает память при останове. Каждый запущенный экземпляр Oracle имеет свою собственную SGA.

SGA включает в себя следующие компоненты, создаваемые в памяти при запуске экземпляра: кэш буфер БД, буфер журнала изменений, разделяемый пул.

Кэш буфер БД хранит последние открытые блоки данных. Эти блоки могут содержать данные, которые изменились, но еще не были записаны на диск (грязные блоки), и данные, которые не изменялись либо были записаны на диск после изменения (чистые блоки). Так как кэш буферов БД хранит блоки данных на основе алгоритма последних используемых блоков, то наиболее активно используемые блоки постоянно остаются в памяти (снижая дисковый ввод-вывод и увеличивая производительность системы).

Буфер журнала изменений хранит данные об изменениях БД, записываясь в файл журнала изменений, используемого для восстановления экземпляра СУБД Oracle в случае сбоя системы, настолько быстро и эффективно, насколько это возможно.

Разделяемый пул хранит такие структуры разделяемой памяти, как разделяемые SQL-области в библиотечном кэше и внутренняя информация словаря данных. Состоит из библиотечного кэша и кэша словаря данных.

Библиотечный кэш используется для хранения разделяемых SQL-выражений. Здесь для каждого уникального SQL-выражения строятся дерево разбора строк и план исполнения, которые кэшируются (т.е. сохраняются в библиотечном кэше). Если несколько приложений отправляют одинаковые SQL-выражения, то для ускорения работы используется разделяемая SQL-область (так как при использовании уже разобранных строк и готового плана исполнения происходит экономия времени).

Кэш словаря данных содержит набор таблиц и представлений, используемых в качестве ссылок к БД Oracle. Здесь хранится информация о логической и физической структуре БД. Словарь данных содержит следующую информацию:

пользовательскую (например, пользовательские привилегии);

ограничения целостности, определенные для таблиц БД;

имена и типы данных всех столбцов таблиц БД;

об объеме памяти, определенном и используемом объектами схемы данных.

Для обеспечения высокой производительности необходимо устанавливать достаточный объем памяти под кэш словаря данных.

• *Программная глобальная область* (PGA) — это область памяти, в которой хранятся данные и управляющая информация о серверных процессах Oracle. Размер и содержание PGA определяют опции, задаваемые при инсталляции Oracle. Эта область включает в себя следующие компоненты:

пространство стека — память, хранящая переменные сеансов, массивы сеансов и т.д.;

информация сеанса (только если Oracle работает не в мультинитиевом режиме);

приватная SQL-область — часть PGA, где хранятся связанные переменные, и буферы реального времени.

Процесс — это механизм выполнения программного кода, который может быть незаметным для пользователя. Кроме того, несколько процессов могут работать одновременно. В разных операционных системах и на разных платформах этот механизм может называться по-разному (процесс, нить, домен и т.д.). В СУРБД Oracle используются два вида процессов: пользовательские процессы и процессы Oracle, также называемые фоновыми, или теневыми. В некоторых операционных системах (например, Windows NT) процессы действительно являются нитями, но, чтобы не путаться в понятиях, будем называть их просто процессами.

- *Пользовательские* (*клиентские*) *процессы* — это пользовательские соединения с СУРБД. Пользовательский процесс управляет вводом и взаимодействует с серверными процессами в Oracle через ее программный интерфейс. Он также используется для выдачи информации пользователю и при необходимости представляет ее в более удобной форме.
- *Процессы Oracle* выполняют функции пользовательских процессов и могут быть разбиты на серверные (выполняющие функции для активных процессов) и фоновые (выполняющие функции СУРБД в целом).

Серверные (теневые) процессы взаимодействуют с процессами пользовательскими и Oracle, исполняя пользовательские запросы. Например, если пользовательский процесс запрашивает часть данных, которых еще нет в SGA, то теневой процесс несет ответственность за считывание блоков данных из БД в SGA. При этом между пользовательским и теневым процессами возникает связь типа «один к одному», хотя один теневой процесс может одновременно взаимодействовать с несколькими пользовательскими (конфигурация мультинитевого сервера), экономя системные ресурсы.

Фоновые процессы используются для выполнения разнообразных задач СУРБД Oracle — от взаимодействия с экземпляром Oracle до записи грязных блоков на диск.

Представим некоторые из фоновых процессов Oracle.

DBWR (DataBase Writer) — ответственен за запись грязных блоков из блоковых буферов БД на диск. Когда транзакция изменяет информацию в блоке данных, этот блок данных не обязан быть немедленно записан на диск. Следовательно, DBWR может записывать данные на диск более эффективно, чем выполнять запись всех изменений по отдельности, т.е. обычно он записывает их тогда, когда они нужны для считывания. Записываются также те данные, которые были недавно использованы.

Для систем с асинхронным вводом-выводом достаточно одного процесса DBWR. Для остальных систем можно значительно увеличить производительность, создав несколько процессов DBWR, таких как LGWR, CKPT, PMON, SMON, RECO, ARCH, LCK*n*.

LGWR (LoG WRiter) — записывает данные из журнального буфера в журнал изменений.

CKPT (CheckK PoinT) — дает сигнал процессам DBWR о необходимости выполнения контрольной точки и обновления всех файлов данных и управляющих файлов. Контрольная точка — это событие, при котором все измененные буферы БД записываются на диск. CKPT — это необязательный процесс. Если процесс CKPT не запущен, его работу принимает на себя процесс LGWR.

PMON (Process MONitor) — используется для поддержания остальных процессов и перезапуска после сбоя, а также очищает неиспользуемые области буферов и освобождает те ресурсы, которые могут быть еще заняты. Ответственен за перезапуск всех зависших процессов и диспетчеров.

SMON (System MONitor) — выполняет восстановление экземпляра при его запуске, что включает очистку временных сегментов и восстановление незаконченных транзакций, а также дефрагментирует БД.

RECO (RECOvery) — очищает незаконченные транзакции в распределенной БД. Выполняет фиксацию или откат спорных транзакций.

ARCH (ARCHiver) — копирует файлы журнала изменений при их заполнении. Активен только в том случае, если СУРБД работает в режиме ARCHIVELOG. При работе системы в других режимах возможны ситуации, в которых не удастся восстановить ее после сбоя.

LCK*n* (Parallel Server LoCK) — использует при работе сервера в параллельном режиме до 10 процессов (*n* — от 0 до 9), которые выполняют функции межэкземплярной блокировки.

5.4. Транзакции

Транзакция — это одна или более SQL-команд, завершаемых фиксацией (commiting) или откатом (rollbacking).

Под *фиксацией* (commiting) понимается принятие и сохранение всех изменений.

Откат (rollbacking) — это процедура отмены последних изменений, т.е. возврат к предыдущему состоянию БД.

Чтобы понять как работает система Oracle, рассмотрим по шагам пример работы простой транзакции.

Итак, транзакция выполняется следующим образом.

1. Приложение обрабатывает пользовательский ввод и создает соединение с сервером посредством SQL*Net.

2. Сервер принимает запрос на соединение и создает серверный процесс.

3. Пользователь выполняет SQL-команду или совокупность команд. (В нашем примере будем считать, что пользователь изменяет данные в строке таблицы.)

4. Серверный процесс просматривает, есть ли в разделяемом пуле SQL-область с идентичными SQL-командами. Если он находит аналогичную разделяемую SQL-область, то серверный процесс проверяет права пользователя на доступ к данным. Предположим, что права имеются, тогда серверный процесс выполняет команды, используя разделяемую SQL-область. Однако если разделяемая SQL-область не найдена, выделяется память под новую область, а затем происходит разбор и выполнение SQL-команд.

5. Серверный процесс ищет данные в SGA (если они там есть) или считывает их из файла данных в SGA.

6. Серверный процесс изменяет данные в SGA. (Запомните, что серверный процесс может только считывать данные из файла данных.) Позже процесс DBWR запишет измененные блоки данных в постоянное хранилище (на жесткий диск, магнитную ленту и т.д.).

7. Пользователь выполняет команду COMMIT (Фиксация) или ROLLBACK (Откат). Первая завершает транзакцию, а вторая отменяет изменения. Если транзакция зафиксирована, процесс LGWR немедленно записывает ее в файл журнала изменений.

8. Если транзакция успешно завершена, клиентскому процессу передается код завершения. Если произошел какой-либо сбой, возвращается сообщение об ошибке.

Примечание. Транзакция не считается зафиксированной до тех пор, пока не завершена запись в файл журнала изменений (redo log file). Этот механизм способствует тому, что в случае сбоя зафиксированная транзакция может быть восстановлена.

5.5. Обеспечение целостности данных

При работе с СУРБД Oracle необходимо решать задачу обеспечения целостности данных (восстановления базы данных после сбоев, перехвата ошибок и т.д.), для чего используются следующие функции: создание контрольных точек, журнализация и архивирование.

Создание контрольных точек (checkpointing). Как уже говорилось, сигнал к созданию контрольной точки поступает либо от процесса DBWR, либо от процесса LGWR. Но что же такое контрольная точка и для чего она необходима?

Так как все изменения блоков данных происходят в блоковых буферах, то изменения данных в памяти не обязательно отража-

ются в этих блоках на диске. Процесс кэширования происходит по алгоритму последнего использованного блока, поэтому буфер, подверженный постоянным изменениям, помечается как последний использованный и процесс DBWR не записывает его на диск. Контрольная точка служит для обеспечения записи этих буферов на диск. Все грязные буферы должны сохраняться на диске в обязательном порядке.

Контрольная точка может работать в двух режимах: нормальной контрольной точки и быстрой контрольной точки.

В режиме *нормальной контрольной точки* грязные буферы записываются последовательно процессом DBWR. Эта контрольная точка выполняется гораздо дольше, чем быстрая, но затрагивает меньше системных ресурсов.

В режиме *быстрой контрольной точки* процесс DBWR записывает одновременно несколько буферов. Эта контрольная точка выполняется очень быстро и более эффективна при вводе-выводе, однако она значительно снижает производительность системы.

Частое выполнение контрольных точек способствует увеличению времени, необходимого на восстановление системы в случае сбоя.

Контрольная точка автоматически выполняется при смене журнала изменений.

Журнализация и архивирование. Журнал изменений (redo log) записывает все изменения БД Oracle. Целью его создания является возможность экстренного восстановления БД в случаях сбоев системы и потери файлов данных. Восстановив файлы данных из ранее сделанных резервных копий, файлы журнала изменений (включая архивные файлы журнала) могут повторить все последние транзакции и таким образом файлы данных будут полностью восстановлены.

Когда файл журнала изменений оказывается полностью заполненным, происходит смена журнала и процесс LGWR заводит новый файл. Во время смены журнала процесс ARCH записывает заполненный файл в архив файлов журнализации. В тот момент, когда архивирование только закончилось, файл журнала изменений помечается как доступный. Очень важно, чтобы архивные файлы журнала изменений надежно хранились, так как они могут понадобиться для восстановления системы.

5.6. Создание триггеров и хранимых процедур

Для создания триггеров, хранимых процедур и просто скриптов (в Oracle их принято называть безымянными блоками) в системе Oracle разработан свой язык, получивший название PL/SQL (Program Language SQL).

Для каждой таблицы можно создать до 12 триггеров. Вот шаблон триггера:

```
CREATE TRIGGER [name] (событие вызова триггера)...
(необязательное ограничение триггера)
BEGIN
  (действие триггера)
  END;
```

При определении триггера можно указать, сколько раз он должен выполняться: для каждой изменяемой строки (row trigger) либо однократно для всего выполняемого выражения независимо от того, сколько строк будет изменено (statement trigger).

ROW TRIGGER — часто используемый вид триггера. Выполняется для каждой строки по одному разу. Например, если SQL-выражение UPDATE обновляет множество строк в таблице, то триггер вызывается для каждой строки, которая изменяется выражением UPDATE.

Если выражение не влияет ни на одну строку, триггер вызываться не будет.

STATEMENT TRIGGER — триггер, вызываемый независимо от числа измененных строк в таблице (и даже если не изменялась ни одна строка). Например, если выражение DELETE удаляет из таблицы несколько строк, триггер уровня этого выражения вызывается только один раз независимо от того, сколько строк удаляется из таблицы.

При определении триггера необходимо указать момент выполнения (trigger timing) тела триггера: до (BEFORE) или после (AFTER) выражения, что применимо как к триггерам выражений, так и к строчным триггерам.

Система Oracle поддерживает еще один вид триггера — INSTEAD-OF (Вместо). Эти триггеры доступны только в редакции Oracle8i.

Они могут использоваться в многотабличных и объектных представлениях.

В отличие от других триггеров они применяются вместо выполнения DML-выражений, т.е. представление можно модифицировать, как обычную таблицу, с помощью выражений INSERT, UPDATE и DELETE, или для соответствующего изменения запустить триггер INSTEAD-OF.

Триггеры INSTEAD-OF активизируются для каждой изменяемой строки.

Контрольные вопросы

1. Что такое триггер?
2. Что такое транзакция?
3. Назовите последовательность выполнения транзакции.

4. Какие категории файлов включает в себя физический уровень БД Oracle?

5. В каком табличном пространстве хранится словарь данных?

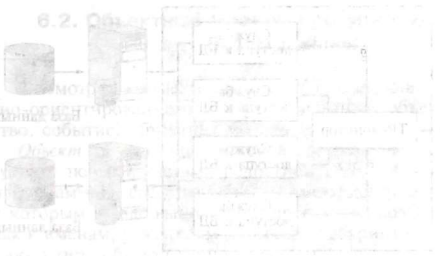
6. Чем различаются сегмент, экстент и блок данных?

7. Какие основные элементы составляют экземпляр Oracle?

8. Для каких целей используется разделяемая память в СУБД Oracle?

9. Каково назначение следующих функций Oracle: создание контрольных точек, журнализация, архивирование?

10. Для каких задач разработан язык PL/SQL?



ГЛАВА 6

ТЕХНОЛОГИИ ДОСТУПА К УДАЛЕННЫМ БАЗАМ ДАННЫХ

6.1. Структура организации доступа к данным в трехуровневой архитектуре

В гл. 1 уже рассматривались концепции организации структур управления базами данных в архитектурах клиент — сервер и файл — сервер.

С развитием информационных технологий, связанных в том числе с необходимостью взаимодействия предприятий через глобальную сеть Интернет, все большее развитие получила трехуровневая схема взаимодействия клиентской и серверной частей (см. рис. 1.7). Как показала практика, эта схема оказалась эффективной и при организации внутренних ЛВС предприятий, где в качестве клиента может использоваться обычный Web-броузер. В соответствии с данной схемой общая структура БД состоит из трех уровней:

1-й — клиенты («тонкие клиенты»);
2-й — сервер приложений;
3-й — сервер базы данных.

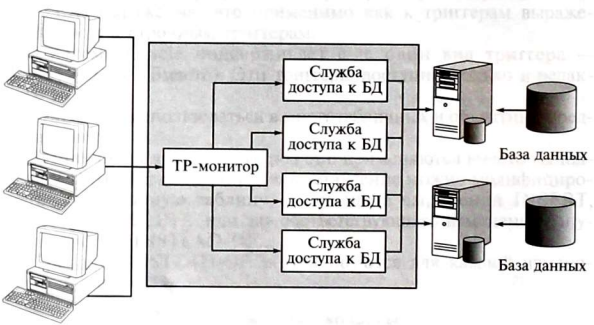


Рис. 6.1. Монитор обработки транзакций

Первый и второй уровни являются прерогативой клиентской части приложения баз данных, а третий — серверной.

Приложение 1-го уровня должно обеспечить пользователю дружественный интерфейс (диалоговые формы) при работе с БД, включая: возможность обращения к базе данных с помощью запросов и получение результатов обращения к базе данных .

Приложение 2-го уровня должно содержать программу, обеспечивающую эффективное выполнение приложения 1-го уровня, включая:

- тексты программ SQL-запросов (транзакций);
- проверку синтаксиса условий выполнения запроса, введенного пользователем;
- программы обеспечения доступа к информации сервера баз данных.

Приложение 3-го уровня должно содержать все таблицы баз данных и обеспечивать эффективное управление работой удаленных пользователей с информацией.

В настоящее время первые две части приложения 2-го уровня разрабатывают с применением так называемых мониторов обработки транзакций ТР-миниторов (рис. 6.1).

Программы обеспечения доступа к информации сервера баз данных разрабатываются с применением различных технологий: ODBC COM; ADO.NET; CORBA; MIDAS; .NET FrameWork.

Все эти технологии основаны на единых принципах — *объектных моделях доступа к удаленным базам данных*, и разрабатываются соответственно на методах объектно-ориентированного программирования.

6.2. Объектные модели доступа к удаленным базам данных

Рассмотрим применительно к СУБД следующие понятия объектно-ориентированного программирования: объект, класс, свойство, событие, объектная модель.

Объект (object) — это типовой программный элемент, используемый любыми частями целостной СУБД, включая аппаратные и программные средства. Объекты имеют дескрипторы (description), к которым в ходе выполнения программы возможно обращение, как к именам, указателям и меткам. Дескрипторы дают информацию о типе объекта и описание характеристик, присущих конкретному объекту. К объектам СУБД относятся таблицы, запросы, формы, отчеты, макросы, модули. Объектами являются также элементы управления, помещаемые в формы, и отчеты.

Класс (class) представляет собой описание совокупности однотипных объектов.

Свойство (property) представляет собой описание характеристики либо отдельного объекта, либо класса объектов.

Событие (event) можно представить как некоторую реакцию объекта на определенные действия пользователя или программы в процессе работы с базой данных.

Объектная модель (object model), или *объектная архитектура* (object architecture), — это совокупность взаимосвязанных объектов, описывающих конкретную программную систему. В таких системах все процессы, связанные с обработкой и управлением информацией, представляются как операции над объектами.

Проблемы оптимизации управления удаленными объектами баз данных связаны с соответствующими алгоритмами (технологиями) доступа к информации.

Как уже говорилось, ядром практически всех реляционных СУБД является язык SQL.

Появление трехуровневых архитектур управления базами данных привело к созданию технологий разработки приложений промежуточного 2-го уровня с применением универсальных языков программирования. В этом случае разработчики вынуждены интегрировать SQL в соответствующие СУБД.

Кроме того, развитие и совершенствование информационных технологий привело к появлению нового направления — объектно-ориентированного проектирования баз данных. Концептуально стратегия объектно-ориентированного проектирования СУБД с применением технологий реляционных баз данных может быть сведена к следующим положениям:

- введение средств работы с базой данных в язык программирования;
- предоставление расширяемых объектно-ориентированных библиотек;
- дополнение языка SQL объектно-ориентированными функциями;
- разработка новых языков и моделей баз данных.

Рассмотрим эти положения.

Введение средств работы с базой данных в существующий объектно-ориентированный язык программирования. При таком подходе традиционные функции базы данных встраиваются в существующие объектно-ориентированные языки программирования, например Smalltalk, C++, Java. Подобный подход используется в языке GemStone, в котором дополняются возможности именно этих трех языков.

Предоставление расширяемых объектно-ориентированных библиотек. При этом подходе также предусматривается введение традиционных функций базы данных в существующий объектно-ориентированный язык программирования. В данном случае вместо расширения функций самого языка используются дополнитель-

ные библиотеки классов, поддерживающие объектные типы данных, транзакции, параллельную обработку, защиту данных и т.д. Этот подход используется в продуктах Ontos, Versant, ObjectStore.

Дополнение языка SQL объектно-ориентированными функциями. Благодаря широкому распространению языка SQL некоторые компании-разработчики пытаются расширить его в целях поддержания объектно-ориентированных конструкций. Этот подход используется компаниями-разработчиками реляционных и объектно-ориентированных СУБД. Поддержка подобных объектно-ориентированных инструментов уже предусматривается в очередной версии стандарта SQL—SQL3.

Разработка новых языков и моделей баз данных. Это наиболее радикальный подход, требующий пересмотра концепций реляционного подхода, с полной ориентацией на объектно-ориентированные модели данных. Необходимость такого подхода связана с специфическими (не реляционными) базами данных, создаваемыми, например, по результатам автоматизированного конструкторского и технологического проектирования с применением систем CAD/CAM.

6.3. Монитор обработки транзакций

Монитор обработки транзакций — это программа, управляющая обменом данных между клиентами и серверами. TP-монитор обеспечивает создание единообразной вычислительной среды, в полной мере отвечающей современным концепциям управления производством посредством создания единого информационного пространства (см. гл. 2).

Применение TP-монитора повышает эффективность управления базами данных за счет:

- маршрутизации транзакций;
- управления распределенными транзакциями;
- уравновешивания нагрузки на серверы;
- мультиплексирования соединений;
- повышения надежности.

Маршрутизация транзакций. TP-монитор позволяет применять средства управления доступа к различным СУБД за счет перенаправления транзакций.

Управление распределенными транзакциями. TP-монитор позволяет управлять транзакциями, которые требуют доступа к данным, хранящимся в нескольких, возможно даже разнородных, СУБД (Oracle, SQL Sever и др.).

TP-мониторы обычно управляют транзакциями с использованием стандарта DTP (Distributed Transaction Processing — обработка распределенных транзакций).

Уравновешивание нагрузки на серверы. ТР-монитор позволяет равномерно распределить клиентские запросы по нескольким СУБД, находящимся на одном или нескольких компьютерах, по принципу перенаправления обращений клиента к службам наименее загруженного сервера. Кроме того, ТР-монитор может для обеспечения необходимого уровня производительности переводить в рабочее состояние дополнительные СУБД.

Мультиплексирование соединений. В среде с большим количеством пользователей иногда возникают сложности обеспечения их одновременного подключения к СУБД. Однако во многих случаях пользователям и не требуется непрерывный доступ к СУБД.

ТР-монитор позволяет перейти от режима, при котором каждый пользователь постоянно подключен к СУБД, к режиму, при котором соединения СУБД устанавливаются только в случае необходимости и поддерживаются лишь до тех пор, пока происходит обмен данными. Кроме того, через одно подобное соединение передаются запросы сразу от нескольких пользователей.

ТР-монитор позволяет предоставить доступ к имеющимся СУБД большему количеству пользователей с использованием меньшего количества соединений, что, в свою очередь, сокращает потребности в ресурсах.

Повышение надежности. ТР-монитор в качестве диспетчера транзакций выполняет все необходимые действия по обеспечению непротиворечивости базы данных, тогда как сервер баз данных действует как диспетчер ресурсов.

В случае отказа СУБД ТР-монитор способен перенаправить транзакцию в другую СУБД или хранить ее в памяти до тех пор, пока работа данной СУБД не восстановится.

ТР-мониторы обычно применяются в среде с очень большим объемом транзакций, в которой они могут снять часть нагрузки с сервера СУБД.

Наиболее распространенными являются ТР-мониторы CICG и Encina компании IBM, которые используются в операционных системах Windows NT.

6.4. Универсальная стратегия доступа к данным ODBC

Как уже говорилось, основная стратегия разработки приложений управления доступом к базам данных заключается в интеграции универсальных языков программирования с языком SQL.

Для выполнения такой интеграции фирма Microsoft разработала универсальный интерфейс, получивший название Open Database Connectivity (ODBC) — открытый доступ к базам данных. Техно-

логия ODBC предусматривает использование единого интерфейса для доступа к разнородным реляционным базам данных. При этом язык SQL рассматривается как стандартное средство доступа к данным.

Интерфейс ODBC, реализованный непосредственно на языке С, обеспечивает высокую степень функциональной совместимости, в результате чего одно и то же приложение может получать доступ к данным, хранящимся в базах различных целевых СУБД, без необходимости внесения изменений в его программный текст. Таким образом, разработчики получили инструмент, позволяющий создавать и распространять приложения с архитектурой клиент—сервер, способные работать с широким спектром различных целевых СУБД.

Для связи приложения с любой выбранной пользователем целевой СУБД достаточно лишь иметь соответствующий драйвер базы данных.

В настоящее время технология ODBC фактически признана как международный стандарт. Основной причиной популярности этой технологии является ее гибкость, предоставляющая разработчикам следующие преимущества:

- приложения больше не связаны с прикладным интерфейсом API (Application Programming Interface) конкретной СУБД;
- операторы SQL могут явно включаться в исходный текст приложения или динамически создаваться непосредственно во время выполнения программы;
- в приложении можно не учитывать особенности используемых протоколов передачи данных;
- данные могут передаваться и приниматься в том формате, который в набольшей степени подходит для данного приложения;
- свойства поддержки разработаны с учетом требований международных стандартов.

Драйверы ODBC разработаны практически для всех реляционных СУБД.

Рассмотрим кратко структуру ODBC, в интерфейс которой включены следующие элементы:

- библиотека функций, вызов которых позволяет приложению подключаться к базе данных, выполнять операторы SQL и извлекать информацию из таблиц баз данных;
- стандартный метод подключения и регистрации в СУБД;
- стандартные средства представления данных различных типов;
- стандартный набор кодов ошибок;
- типовой синтаксис операторов SQL.

Архитектура ODBC состоит из четырех компонентов: приложения, диспетчера драйверов, драйверов и агентов баз данных, источников данных.

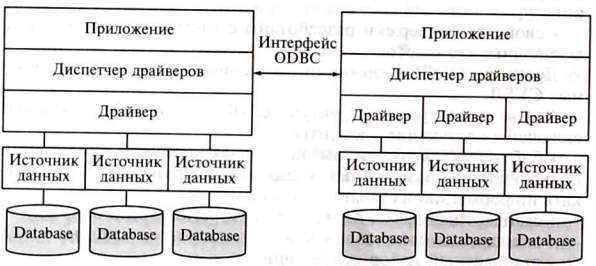
Приложение выполняет обработку данных и вызов функций библиотеки ODBC для отправки операторов SQL в СУБД и выборки информации из таблиц баз данных.

Диспетчер драйверов выполняет загрузку и выгрузку драйверов в соответствии с алгоритмом работы приложения. Этот программный компонент может сам обрабатывать вызовы функций или передавать их драйверу. Разработан компанией Microsoft и представляет собой динамически связываемую библиотеку DLL (Dynamic Link Library).

Драйверы и агенты баз данных обрабатывают вызовы функций ODBC, направляют запросы SQL в конкретные источники данных и возвращают полученные результаты приложению. При необходимости драйверы модифицируют исходный запрос приложения для приведения его в соответствие синтаксическим требованиям целевой СУБД.

Драйверы могут представлять только возможности, обеспечиваемые целевой СУБД, т.е. от них не требуется собственной реализации возможностей, которые не поддерживает СУБД. Например, если целевая СУБД не поддерживает операции внешнего соединения, то эта функция не будет поддерживаться и драйвером ODBC. Единственным исключением из этого правила являются драйверы для СУБД, не имеющих собственных машин баз данных, например XBase. В этом случае драйвер реализует машину базы данных, которая поддерживает минимальный набор операторов SQL.

Существует две схемы реализации ODBC: с использованием одного и нескольких драйверов (рис. 6.2).



а *б*

Рис. 6.2. Схемы реализации ODBC с использованием одного (*а*) и нескольких (*б*) драйверов

В варианте архитектуры ODBC с использованием единственного драйвера для каждого типа СУБД потребуется применение агентов базы данных, размещаемых в серверной части приложения. При обработке запросов на доступ к базе данных эти агенты взаимодействуют с драйвером ODBC, расположенным в клиентской части приложения.

В варианте архитектуры ODBC с использованием нескольких драйверов все указанные ранее задачи должны решаться каждым драйвером ODBC, и в этом случае не требуется применения агента базы данных.

В среде Windows драйвер реализован в виде библиотеки DLL. Агенты баз данных реализуются как процессы, которые функционируют на сервере с установленной целевой базой данных.

Источники данных содержат те данные, доступ к которым необходим пользователю приложения, и сохраняются они в базе данных контролируемой целевой СУБД и операционной системой.

Несмотря на то что технология ODBC является практически стандартным средством доступа к данным, она имеет много ограничений, в том числе связанных с доступом к объектам нереляционных баз данных.

Дальнейшим развитием технологий доступа к различным данным, как реляционным, так и нереляционным (файловым и почтовым системам, графическим и мультимедийным объектам), является технология OLE DB.

Технология OLE DB (Object Linking and Embedding for DataBases), созданная фирмой Microsoft, — это объектно-ориентированная система, разработанная на основе C++ API.

Учитывая большой потребительский спрос на разработку корпоративных информационных систем, разработчики программных продуктов постоянно совершенствуют имеющиеся и создают новые технологии доступа к удаленным базам данных. Рассмотрим наиболее широко применяемые из них в настоящее время: COM; ADO .NET; .NET FrameWork; CORBA; MIDAS.

6.5. Технологии COM

Технология доступа к удаленным данным Component Object Model (COM) — компонентная модель объектов, разработанная фирмой Microsoft как средство взаимодействия приложений (в том числе составных частей операционной системы Windows), функционирующих на одном компьютере.

В дальнейшем технология COM усовершенствовалась для управления объектами базы данных, расположенных в пределах локальной вычислительной сети.

На технологии COM построены такие методы управления удаленными объектами, как OLE, Автоматизация, ActiveX.

• *Метод OLE* (Object Linking and Embedding) — связывание и объединение объектов — протокол, обеспечивающий обмен данными между приложениями. С помощью OLE пользователи могут связывать или внедрять объекты различных приложений (в том числе и баз данных) в файлы других приложений. (Одним из типов полей в реляционных базах данных является OLE.) Каждый объект OLE характеризуется двумя компонентами: собственно информацией, содержащейся в исходном файле, и адресом нахождения файла на дисковом пространстве компьютера или адресом файла в локальной вычислительной сети.

Развитием технологии OLE является технология OLE DB — программный интерфейс, удовлетворяющий структуре COM и предоставляющий унифицированный способ доступа к различным файлам, в общем случае не являющимся базами данных. При этом объекты управления данными ADO (ActiveX Data Objects) являются промежуточным звеном между серверной и клиентской частями баз данных.

• *Метод Автоматизация*, называемый иногда автоматизацией OLE, обеспечивает взаимодействие клиентских и серверных приложений программным способом, например с применением языка VBA.

• *Метод ActiveX* является 32-разрядной версией элементов управления OLE.

Таким образом, технология COM представляет собой различные методы управления удаленными объектами баз данных, построенных в архитектуре типа клиент—сервер, в том числе предназначенных и для работы в ЛВС.

6.6. Технологии ADO .NET

Технология доступа к удаленным базам данных ADO .NET была разработана также для архитектуры клиент—сервер. Однако все возрастающая сложность систем обработки информации потребовала качественного изменения этой архитектуры. Кроме двух уровней удаленных баз данных — клиентского и серверного — появляются дополнительные уровни — серверы бизнес-логики, реализующие бизнес-логику приложений (см. рис. 1.7).

Технология ADO .NET устанавливает следующую схему работы клиента с сервером баз данных:

- установка соединения с сервером;
- получение необходимых данных;
- закрытие соединения;
- обработка данных;

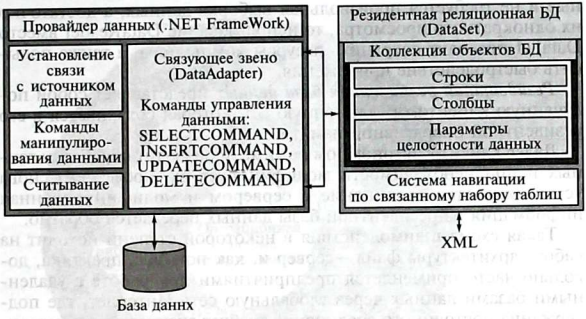


Рис. 6.3. Структура технологии ADO .NET

• установка соединения для передачи измененных данных обратно на сервер.

Реализация данной схемы работы клиента определяется структурой ADO .NET, показанной на рис. 6.3.

Основу ADO .NET составляют два основных модуля: Провайдер данных (Data Provider .NET FrameWork) и Резидентная реляционная база данных (DataSet).

Провайдер данных, как это следует из его названия, отвечает за связь приложения с источником данных и за манипуляцию данными. Провайдер данных включает в себя следующие объекты манипулирования данными: Connection, Command, DataAdapter, DataReader.

Рассмотрим кратко назначение этих объектов.

Connection используется для установления соединения с источником данных, а также для управления транзакциями.

Command позволяет манипулировать данными источника, а также выполнять хранимые процедуры. При этом могут использоваться параметры для передачи данных в обоих направлениях.

DataAdapter служит связующим звеном между резидентной БД DataSet и источником данных и использует обычно объект Command для выполнения команд SQL как при заполнении DataSet данными, так и при обратной передаче измененных клиентом данных к источнику. Для выполнения этих функций в нем имеются четыре метода: SelectCommand, InsertCommand, UpdateCommand и DeleteCommand.

DataReader обеспечивает получение данных от источника только для считывания. Если приложение клиента не модифицирует дан-

ные и не требуется произвольная выборка данных, а достаточно их однократного просмотра, то использование DataReader вместо DataSet позволит сохранить ресурсы компьютера, а также повысить быстродействие приложения.

Резидентная реляционная база данных представляет собой полученную клиентом реляционную БД, которая сохраняется в его резидентной оперативной памяти.

Далее клиент в автономном режиме производит обработку данных и при необходимости модифицирует их, после чего снова устанавливается соединение с сервером и модифицированная информация из резидентной базы данных передается обратно.

Такая схема взаимодействия в некоторой степени походит на работу архитектуры файл—сервер и, как показала практика, довольно часто применяется предприятиями при работе с удаленными базами данных через глобальную сеть Интернет, где поддержание постоянного соединения требует значительных материальных затрат.

Для обеспечения доступа к объектам через глобальную сеть Интернет, которые, как правило, разрабатываются в форматах HTML и XML, в составе ADO .NET и был предусмотрен модуль .NET FrameWork, обеспечивающий взаимодействие между различными форматами представления данных, в том числе HTML и XML.

Из указанных характеристик видно, что технология ADO .NET обеспечивает:

- возможность взаимодействия между данными различных форматов, в том числе HTML и XML;
- значительное снижение затрат при работе с удаленными базами данных через глобальную сеть Интернет.

6.7. Технологии .NET FrameWork

Технология .NET FrameWork, или провайдер данных .NET FrameWork, как она была названа при описании ADO.NET, обеспечивает согласование объектно-ориентированной среды программирования для выполнения кода программы как в рамках ЛВС предприятия, так и в среде Интернет.

На рис. 6.4 показана структура провайдера данных .NET FrameWork, которая включает в себя: среду выполнения (ASP.NET); библиотеку классов; библиотеки настраиваемых объектов; управляемые приложения, в том числе веб-приложения; неуправляемые приложения.

Среда выполнения. Основным свойством среды выполнения является возможность управления кодом программы, сформированной различными языками программирования, поэтому ее также называют общеязыковой средой выполнения.

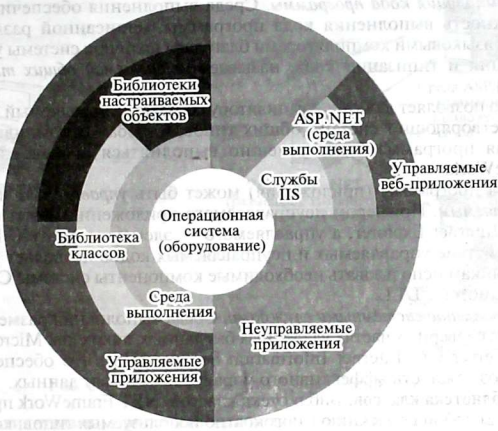


Рис. 6.4. Структура провайдера данных .NET FrameWork

Общеязыковая среда выполнения обеспечивает управление памятью ЭВМ, обеспечение безопасности кода, компиляцию кода программы, управление системными службами.

Эти средства являются внутренними для управляемого кода в общеязыковой среде выполнения.

Управление памятью ЭВМ. Среда выполнения предназначена для повышения производительности процессов обработки информации при работе с удаленными базами данных. Повышение производительности осуществляется оптимизацией работы с памятью ЭВМ так называемым *диспетчером памяти*, который при необходимости увеличивает объем адресуемой памяти. В этом случае среда выполнения освобождает память за счет удаления ссылок на объекты, которые не используются программой.

Обеспечение безопасности кода программы. Для обеспечения защиты от несанкционированного доступа к базам данных управляемым компонентам приложений присваивают разные условия безопасности, т.е. задают условия, при которых управляемый компонент может или не может выполнять операции доступа к файлам, или другие функции, связанные с обработкой и управлением информацией. Например, пользователи могут доверять исполняемой программе, внедренной в веб-страницу, воспроизведение анимации на экране или звукозаписи, не допуская при этом доступ к личным данным, файловой системе или сети.

Компиляция кода программы. Среда выполнения обеспечивает надежность выполнения кода программы, написанной различными языковыми компиляторами благодаря наличию системы унификации и типизации кода, называемой *системой общих типов* (CTS).

Это позволяет любому компилятору создавать управляемый код, удовлетворяющий системе общих типов, а любая откомпилированная программа будет успешно выполняться в среде .NET FrameWork.

Код программы (приложения) может быть *управляемым* и *неуправляемым*. Примером неуправляемого приложения может служить Internet Explorer, а управляемого — элементы ActiveX. Взаимодействие управляемых и неуправляемых кодов позволяет разработчикам использовать необходимые компоненты системы COM и библиотеки DLL.

Управление системными службами. Среда выполнения размещается на серверных частях удаленных баз данных, таких как Microsoft SQL Server или Internet Information Services (IIS), что обеспечивает возможность эффективного управления базами данных.

Библиотека классов. Библиотека классов .NET FrameWork представляет собой коллекцию многократно используемых типов, которые надежно интегрируются с общеязыковой средой выполнения.

Библиотека классов является объектно-ориентированной.

Классы .NET FrameWork позволяют выполнять ряд следующих общих задач программирования: управление строками, сбор данных, подключение к базам данных и доступ к файлам.

В дополнение к этим задачам библиотека классов включает в себя типы, позволяющие использовать .NET FrameWork для разработки текстовых приложений, графических пользовательских интерфейсов (GUI) Windows (Windows Forms), приложений ASP .NET, веб-служб XML и служб Windows.

Например, классы Windows Forms представляют собой набор многократно используемых типов, существенно упрощающих разработку графических интерфейсов пользователя Windows.

Разработка управляемых и неуправляемых приложений. Среда выполнения .NET FrameWork позволяет эффективно разрабатывать клиентские и серверные приложения управления удаленными базами данных.

Клиентские приложения по стилю ближе всего к обычным приложениям в программировании для Windows. Такие приложения открывают на рабочем столе окна или формы пользовательского интерфейса, например при работе с базами данных.

В число клиентских входят также стандартные приложения Microsoft Office (текстовые редакторы, электронные таблицы и др.).

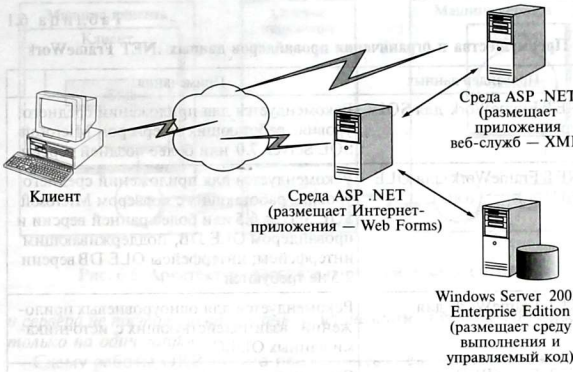


Рис. 6.5. Базовая схема сети с управляемым кодом

Клиентскими приложениями являются элементы ActiveX и элементы управляемых Windows Forms, развертываемых через Интернет как веб-страницы. Такие приложения можно разрабатывать с помощью универсальных языков программирования, однако система .NET FrameWork объединяет возможности языковых средств в единую согласованную среду, существенно упрощающую разработку клиентских приложений.

Одним из достоинств системы .NET FrameWork является разработка *серверных приложений* для работы с базами данных, находящимися в глобальной сети Интернет.

На рис. 6.5 показана базовая схема сети с управляемым кодом, который выполняется в среде разных серверов. Например, на Интернет-серверах Web Forms и XML выполняются стандартные операции по работе с объектами базы данных, тогда как бизнес-логика приложения реализуется через управляемый код на сервере бизнес-логики Windows Server 2003.

Таким образом, завершая ознакомление с технологией .NET FrameWork, еще раз обратим внимание на то, что ее провайдер данных является промежуточным уровнем между клиентской и серверной частями систем управления удаленными базами данных.

В зависимости от типов серверов управления базами данных разработаны соответствующие версии .NET FrameWork.

В табл. 6.1 приведены некоторые особенности провайдеров данных .NET FrameWork, используемые в разных серверных программах.

Таблица 6.1

Преимущества и ограничения провайдеров данных .NET FrameWork

Провайдер данных	Примечания
.NET FrameWork для SQL-сервера	Рекомендуется для приложений среднего уровня, работающих с сервером Microsoft SQL Server 7.0 или более поздней версии
.NET FrameWork для OLE DB	Рекомендуется для приложений среднего уровня, работающих с сервером Microsoft SQL Server 6.5 или более ранней версии и провайдером OLE DB, поддерживающим интерфейсы; интерфейсы OLE DB версии 2.5 не требуются
.NET FrameWork для ODBC	Рекомендуется для одноуровневых приложений, взаимодействующих с источниками данных ODBC
.NET FrameWork для Oracle	Рекомендуется для одноуровневых приложений, взаимодействующих с источниками данных Oracle.Поддерживает клиентское программное обеспечение 8.1.7 или более поздней версии

6.8. Технологии CORBA

Архитектура распределенной системы CORBA. Технология удаленного доступа к базам данных CORBA (Common Object Regust Broker Architecture — общая архитектура объектных заявок) представляет собой промежуточное программное обеспечение, устанавливающее отношения клиент—сервер между объектами в распределенной компьютерной среде.

На рис. 6.6 представлена типовая архитектура распределенной системы CORBA, которая включает в себя следующие компоненты:

- ORB (Object Request Broker) — брокер объектных запросов (заявок), включающий в себя язык IDL;
- IDL (Interface Definition Language) — язык определения интерфейсов;
- POA (Portable Object Adapter) — адаптер объектов;
- Stub — заглушка;
- Skeleton — основа;
- Smart Agent — «умный» агент.

Брокер объектных запросов. ORB устанавливает отношения клиент—сервер между объектами. В данной технологии *роли клиента*

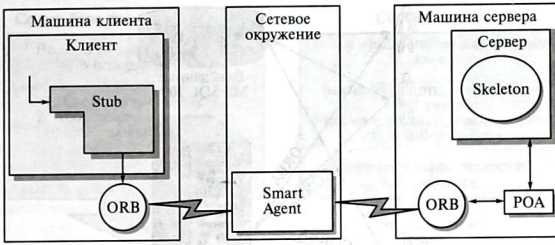


Рис. 6.6. Архитектура распределенной системы CORBA

и сервера не постоянно приписаны компонентам, а устанавливаются только на один запрос.

Схему работы ORB можно представить следующим образом:

- клиент (объект-отправитель) вызывает операцию (метод) на адресате, идентифицируемом ссылкой, и передает запрос ORB;
- посредник по ссылке находит сервер, содержащий объект-адресат (в терминологии CORBA этот сервер иногда называют object implementation — реализация объекта), и активизирует его. Затем доставляет запрос к объекту-адресату, передает адресату параметры, вызывает соответствующую операцию и возвращает результат клиенту после выполнения операции.

В одном запросе могут быть указаны несколько адресатов, которые могут располагаться как на одном, так и на разных серверах.

Система CORBA позволяет организовывать различные архитектурные схемы ORB, однако каждая из них должна реализовывать три категории операций:

- одинаковые для всех реализаций ORB;
- специфичные для конкретного объектного типа;
- специфичные для отдельных видов реализаций объектов.

Возможны следующие схемы брокера объектных запросов:

- ORB, включаемый в клиентское и серверное приложения;
- ORB, выполненный в виде сервера;
- ORB, реализованный как часть операционной системы;
- ORB, основанный на библиотеках.

ORB, включаемый в клиентское и серверное приложения, формируется в виде набора подпрограмм, выполняемых как на стороне клиента, так и на стороне сервера.

ORB, выполненный в виде сервера, реализуется в виде отдельного приложения, что позволяет обеспечить централизованную обработку информации. На рис. 6.7 показана схема такой реализации системы CORBA.

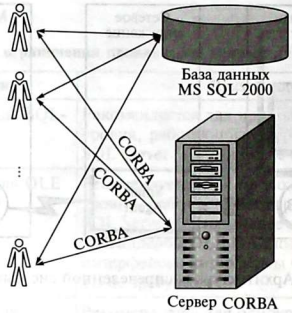


Рис. 6.7. Схема CORBA-сервер

На практике схему реализации ORB в виде сервера применяют для размещения на нем нереляционных баз данных. Так, для примера на рис. 6.8 показана комбинированная схема организации удаленного доступа к данным в геоинформационной картографической системе.

На сервере базы данных MS SQL хранится информация в виде реляционных таблиц — дискретных (параметрических) характеристик карт местности (поверхностей). Доступ к информации, хранящейся на этом сервере, осуществляется по технологии ADO.

На сервере базы данных CORBA-сервер хранятся карты местности в виде отдельных графических файлов, доступ к которым осуществляется по технологии CORBA.

ORB, реализованный как часть операционной системы, используется для повышения надежности, защиты данных и достижения лучшей производительности. В этом случае возможно применение различных методов оптимизации обработки данных, например отмены кодирования и декодирования данных, если клиентская и серверная части находятся на одном компьютере.

ORB, основанный на библиотеках, используется, если код объекта занимает небольшой объем и не требует никаких дополнительных средств.

Язык определения интерфейсов. IDL — это специальный язык для описания ORB и других компонентов системы CORBA, который не содержит присваиваний, операторов if или while, функций и логических переходов. IDL представляет собой процедуры описания, декларации, пассивные определения атрибутов, родительских классов, типов поддерживаемых событий, методов (включая входные и выходные данные), основные и составные типы данных, исключительные ситуации для обработки ошибок.

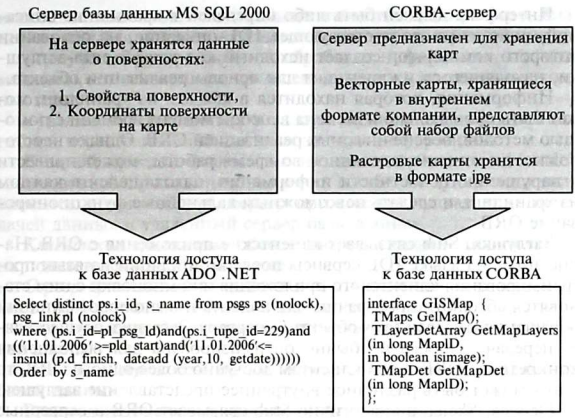


Рис. 6.8. Комбинированная схема организации удаленного доступа к данным с использованием CORBA

IDL описывает интерфейсы аналогично классам в C++ или в Smalltalk, интерфейсы в Java, пакеты в Ada95. Каждый интерфейс определяет операции, которые могут быть вызваны клиентами. Синтаксически IDL является подмножеством C++ с дополнительными ключевыми словами. Существуют компиляторы IDL в C++, Java, Ada, Smalltalk, COBOL, OLE (Visual Basic, PowerBuilder, Delphi).

Одна из задач IDL — обеспечение взаимодействия через ORB всех приложений клиент—сервер.

В связи с этим систему CORBA называют стандартом для разработки промежуточного программного обеспечения, которое поддерживает работу с удаленными базами данных.

Адаптер объектов. POA (в первых версиях CORBA — BOA — Basic Object Adapter) — это первичный путь для обеспечения сервиса конкретной реализацией объекта. Предполагается, что имеется несколько адаптеров объектов, каждый из которых обеспечивает доступ к объектам определенного вида.

Сервисы, которые обеспечиваются ORB посредством адаптеров объектов, часто включают в себя: генерацию и интерпретацию ссылок на объекты, вызов методов, активацию и деактивацию реализаций объектов, а также регистрацию конкретных реализаций и отображение объектных ссылок и реализаций.

Интерфейс должен быть либо определен в хранилище описаний, либо иметь соответствующее IDL-описание, на основании которого компилятор создает исходный код для объекта-заглушки, находящегося у клиента, и для основы реализации объекта.

Информация, которая находится в каждом из хранилищ, может быть произвольно изменена в любой момент времени с помощью методов, обеспечиваемых реализацией ORB. Однако неосторожное изменение, сделанное во время работы, может привести к нарушению целостности информации, находящейся в каждом из хранилищ, и сделать невозможным дальнейшее функционирование ORB.

Заглушка. Stub связывает клиентские приложения с ORB. Написанные на языке IDL сервисы после компиляции на язык программирования клиентского приложения и компоновки с ним становятся абсолютно прозрачны для клиента и выполняют функции обращения к удаленному объекту — серверу. Заглушка осуществляет передачу запроса и обычно оптимизирована для выполнения конкретного ORB. Если клиентам доступно более одного ORB, то у них может быть различное внутреннее представление заглушек.

Основа. Skeleton аналогично Stub связывает ORB и серверные приложения. Создается после компиляции языка IDL. Интерфейс динамического вызова — DII (Dynamic Invocation Interface) — позволяет объекту создавать запрос в реальном времени. Структура запроса, его параметры и атрибуты и даже сама ссылка на объект-адресат генерируются в DII либо на основе запроса к объекту и анализа отклика.

«Умный» агент» — это динамический сервис, моделирующий сетевой каталог, в котором зарегистрированы известные ему серверы объектов как в локальной сети, так и в сети Интернет. Он отыскивает требуемый сетевой адрес сервера и передает ему запрос ORB.

6.9. Технологии MIDAS

Технология MIDAS (Multitier Distributed Applications Services) — набор сервисов для создания многозвенных распределенных приложений.

Многозвенное приложение представляет собой распределенные системы удаленного доступа к данным, которые состоят, как минимум, из трех логических уровней. Эти логические уровни могут находиться как на одном, так и на нескольких компьютерах.

Применение многозвенных приложений позволяет обеспечить следующие преимущества:

• формирование пакета бизнес-логики в общедоступном среднем уровне, доступ на который могут получить одновременно сразу

несколько клиентов, что позволит избежать дублирования бизнес-логики для каждого отдельного клиентского приложения;

- получение распределенной обработки информации, т.е. возможность оптимизации распределения нагрузки на отдельные компьютеры;
- увеличение устойчивости за счет возможности организации гибкой перестраиваемой системы защиты информации.

В самой простой форме (так называемой three-tiered model) многозвенное приложение включает в себя следующие уровни: клиентское приложение, сервер приложений, управление передачей данных и удаленный сервер базы данных.

Клиентское приложение обеспечивает интерфейс пользователя на пользовательском компьютере.

Сервер приложений находится в доступном для всех клиентов месте и обеспечивает общую передачу данных.

Управление передачей данных обеспечивает так называемый брокер данных.

Удаленный сервер базы данных обеспечивает систему управления базой данных.

Взаимодействие указанных уровней осуществляется следующим образом.

1. Пользователь запускает клиентское приложение.
2. Клиент соединяется с сервером приложений (который может определяться как во время исполнения, так и во время создания приложения).
3. Запускается сервер приложений.
4. Клиент получает интерфейс IAppServer от сервера приложений.
5. Клиент запрашивает данные из сервера приложений, который, в свою очередь, запрашивает информацию из базы данных, упаковывает ее для клиента и возвращает пакет данных клиенту.
6. Клиент расшифровывает пакеты данных и предоставляет их пользователю.
7. Пользователь взаимодействует с клиентским приложением. При изменении данных клиент упаковывает измененные данные в пакеты и отсылает их на сервер приложений.
8. Сервер приложений расшифровывает пакеты и сохраняет изменения в контексте транзакции. Если запись не может быть сохранена на сервере, последний пытается согласовать изменения с текущими данными и отделяет данные, которые не могут быть сохранены. Когда процесс обработки измененных данных закончен, сервер возвращает все несохраненные данные клиенту для дальнейшего уточнения.
9. Клиент уточняет необработанные данные, после чего посылает их снова серверу приложений. Затем клиент обновляет свои данные с сервером.

Разработка пользовательских приложений производится с применением языка программирования Delphi.

Технология MIDAS позволяет:

- получать доступ к данным, физически расположенным на разных машинах;
- распределять нагрузку ресурсов по сети, что позволяет уменьшить сетевой трафик;
- разделить бизнес-логику приложения на отдельные части, что повышает уровень безопасности работы с базами данных.

Эффективную разработку приложений MIDAS обеспечивают следующие основные компоненты:

- модули удаленных данных;
- компонент TClientDataSet набора данных клиента;
- компоненты связи TDCOMConnection, TSocketConnection, TWebConnection, TCorbaConnection;
- брокер бизнес-объектов SimpleObjectBroker.

Модули удаленных данных — это специальные модули данных, которые действуют как серверы автоматизации или как CORBA-серверы, предоставляя клиентам доступ к любым провайдерам, которые они содержат. Используются на сервере приложений.

Компонент набора данных клиента TClientDataSet — это специализированный набор данных, который использует MIDAS .DLL для управления.

Компоненты связи TDCOMConnection, TSocketConnection, TWebConnection, TCorbaConnection — это набор компонентов, которые определяют сервер приложений, тип взаимодействия между клиентом и сервером и формируют интерфейс, доступный для наборов данных клиента. Каждый из этих компонентов специализируется на конкретном протоколе связи.

Брокер бизнес-объектов SimpleObjectBroker служит для распределения вычислительной нагрузки по нескольким серверам.

С помощью технологии MIDAS можно создавать системы, которые могут обрабатывать запросы Интернет-приложений. MIDAS работает одинаково хорошо с технологиями CORBA, COM, OLEnterprise и MTS и упрощает интеграцию существующих систем.

Контрольные вопросы

1. В чем состоит принципиальное отличие двухуровневой клиент-серверной схемы организации баз данных от трехуровневой?
2. Что означают понятия «объектная модель» и «объектная архитектура»?
3. Какие задачи выполняет программа *Монитор обработки транзакций*?
4. Каковы основные причины популярности системы доступа к удаленным базам данных?
5. Рекомендуется ли применять технологию удаленного доступа к данным в сети Интернет?

6. Для решения каких задач применяют технологию удаленного доступа к данным ADO .NET?

7. Поясните схему работы клиента с сервером баз данных в технологии ADO .NET.

8. Какая технология доступа к данным в среде Интернет была разработана в ADO .NET?

9. Из каких компонентов состоит среда .NET FrameWork?

10. Для каких целей применяют технологию доступа к данным CORBA?

11. Каково назначение следующих компонентов системы CORBA: ORB, IDL, POA, Stub, Skeleton, Smart Agent?

12. Для чего рекомендуется применять технологии доступа к данным MIDAS?

ЧАСТЬ III

ПРОЕКТИРОВАНИЕ СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ БАЗ ДАННЫХ

ГЛАВА 7

МЕТОДИЧЕСКИЕ ОСНОВЫ ПРОЕКТИРОВАНИЯ СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ

Методология разработки серверной части приложения предусматривает разбиение всего процесса проектирования на концептуальное, логическое и физическое.

Концептуальное проектирование баз данных должно отражать единую информационную модель предприятия, не зависящую от программных и технических условий реализации информационной системы.

Концептуальное проектирование базы данных включает в себя следующие этапы.

1. Создание локальной концептуальной модели данных.
2. Определение типов сущностей.
3. Определение атрибутов.
4. Определение типов связей.
5. Проверка модели на избыточность.
6. Проверка соответствия локальной концептуальной модели конкретным пользовательским транзакциям.
7. Обсуждение локальных концептуальных моделей данных с конечными пользователями.

Создание локальной концептуальной модели данных. Целью данного этапа является определение предметной области и состава пользователей разрабатываемой базы данных.

На рис. 7.1 показан пример локальной концептуальной модели, устанавливающей предметную область и связи между подразделениями предприятия в процессе изменения конструкторской документации в разработанной базе данных «Извещение».

Примечание. База данных «Извещение» была разработана для информационной поддержки процесса принятия решений на изменение конструкторской документации (КД) в процессе жизненного цикла изделий, а также для автоматизированного составления извещений на эти

Рис. 7.1. Структура связей между подразделениями приборостроительного предприятия

изменения. Предметной областью рассматриваемой базы данных является создаваемый документ «Извещение на изменение конструкторской документации».

Определение типов сущностей. Цель данного этапа сводится к установлению оптимального состава таблиц базы данных в соответствии с задачами каждого конкретного пользователя и с учетом принципов нормализации.

Определение атрибутов. На данном этапе для каждой таблицы БД устанавливается необходимый состав атрибутов (признаков), описывающий конкретную сущность, а также необходимые ключевые поля.

Определение типов связей. На данном этапе устанавливаются связи между таблицами баз данных в целях обеспечения целостности информации при различных ее модификациях.

На рис. 7.2 показан пример, отражающий типы сущностей, и соответствующий состав таблиц и связей между ними в базе данных «Извещение».

Проверка модели на избыточность. На данном этапе база данных проверяется на избыточность связей и производится ее устранение в случае обнаружения.

Связь является избыточной, если представленная в таблице информация может быть получена из другой таблицы базы дан-

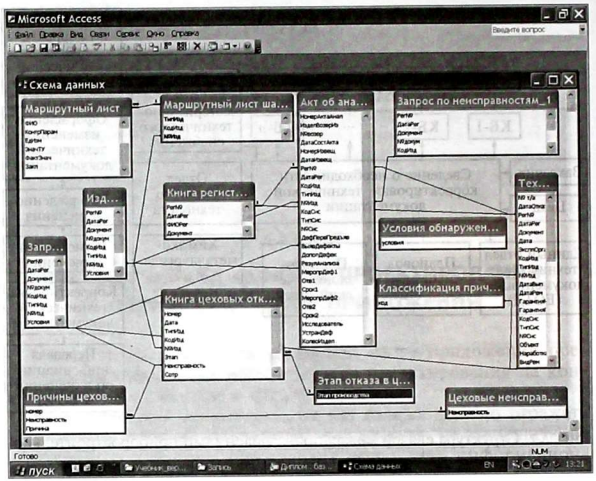


Рис. 7.2. Состав таблиц и связи между ними в базе данных «Извещение»

ных. Для устранения таких ситуаций проводится анализ таблиц, имеющих связи типа «один к одному».

Проверка соответствия локальной концептуальной модели конкретным пользовательским транзакциям. На данном этапе проверяется полнота получения информации каждым пользователем базы данных в результате выполнения пользовательских запросов или транзакций.

Обсуждение локальных концептуальных моделей данных с конечными пользователями. Очевидно, что данный этап необходим для подтверждения того, что разработанная модель базы данных полностью удовлетворяет требованиям конечных пользователей базы данных.

Логическое проектирование баз данных должно отражать непосредственные связи между пользователями информации, обеспечивающие целостность данных в процессе эксплуатации единого информационного пространства. На данном этапе необходимо учитывать выбранную для реализации конкретную СУБД.

Логическое проектирование базы данных (для реляционной модели) включает в себя следующие этапы.

1. Создание и проверка локальной логической модели данных на основе представления предметной области каждого из типов пользователей.

2. Устранение особенностей локальной логической модели, несовместимых с реляционной моделью (необязательный этап).

3. Определение набора отношений исходя из структуры локальной логической модели данных.

4. Проверка отношений с помощью правил нормализации таблиц.

5. Проверка соответствия отношений требованиям пользовательских транзакций.

6. Определение требований поддержки целостности данных.

7. Обсуждение разработанных локальных логических моделей данных с конечными пользователями.

8. Создание и проверка глобальной логической модели данных.

9. Слияние локальных логических моделей данных в единую глобальную модель данных.

10. Проверка глобальной логической модели данных.

11. Проверка возможностей расширения модели в будущем.

12. Обсуждение глобальной логической модели данных с пользователями.

На рис. 7.3 показана схема реализации базы данных «Извещение» как элемента глобальной системы в условиях внедрения на предприятии единого информационного пространства на основе системы SQL Server2000 и системы электронного документооборота PartY Plus.

Физическое проектирование базы данных предусматривает принятие разработчиками окончательного решения о способах реализации создаваемой базы данных в условиях применения конкретной СУБД.

Физическое проектирование базы данных (для реляционной модели) включает в себя следующие этапы.

1. Перенос глобальной логической модели данных в среду целевой СУБД.

2. Проектирование базовых отношений в среде целевой СУБД.

3. Проектирование отношений, содержащих производные данные.

4. Реализация ограничений предметной области.

5. Проектирование физического представления базы данных.

6. Анализ транзакций.

7. Выбор файловой структуры.

8. Определение индексов.

9. Определение требований к дисковой памяти.

10. Разработка пользовательских представлений.

11. Разработка механизмов защиты.

12. Анализ необходимости введения контролируемой избыточности.

13. Организация мониторинга и настройка функционирования операционной системы.

Рис. 7.3. Схема реализации базы данных «Извещение» в системе электронного документооборота

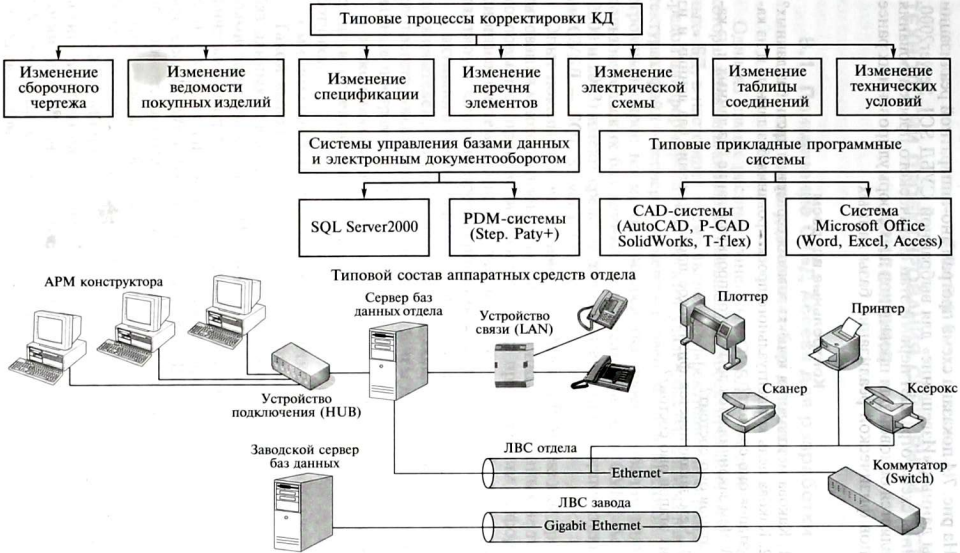


Рис. 7.4. Схема программно-аппаратной реализации базы данных «Извещение»

На рис. 7.4 показана схема программно-аппаратной реализации базы данных «Извещение» для выбранной СУБД SQL Server2000. Данная схема отражает первый этап физического проектирования и является основой для проведения всех сформулированных ранее этапов физической реализации базы данных.

Контрольные вопросы

1. Какова методология проектирования серверной части баз данных?
2. Какова цель концептуального проектирования баз данных и из каких этапов оно состоит?
3. Чем заканчивается логическое проектирование баз данных и из каких этапов оно состоит?
4. Чем заканчивается физическое проектирование баз данных и из каких этапов оно состоит?

ГЛАВА 8

ТЕХНОЛОГИИ ПРОЕКТИРОВАНИЯ СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ

8.1. Применение СУБД Access для разработки проекта удаленных баз данных

Одна из самых распространенных СУБД — Microsoft Access, которая изначально разрабатывалась для проектирования «настольных» БД, в настоящее время является эффективной программной средой для формирования проектов удаленных баз данных в архитектурах файл—сервер и клиент—сервер. Фирма Microsoft учла огромный спрос на методы и средства проектирования многопользовательских баз данных, и последние версии СУБД Access (2000 и 2003) имеют средства для ее интеграции с корпоративными СУБД — SQL Server и Oracle.

Опыт показывает, что применение Microsoft Access для проектирования многопользовательских удаленных баз данных не только повышает производительность труда при создании информационных систем, но и, что очень важно, в большей степени гарантирует качество проектирования за счет:

- активного привлечения к разработке приложений конечных пользователей, которые, несомненно, более квалифицированы в соответствующей предметной области, чем высококвалифицированные программисты;
- максимального использования визуальных (диалоговых) средств проектирования, сводящих к минимуму ошибки разработчика;
- минимальной трудоемкости разработки проекта базы данных.

Такой вывод основан на том, что для создания компонентов баз данных (таблиц, запросов, форм и отчетов) самими конечными пользователями в Microsoft Access разработан простой графический интерфейс пользователя (Graphical User Interface — GUI). Кроме того, в данной системе имеются программы, называемые мастерами (Wisards), и конструкторы (Builders), позволяющие пользователю в режиме запрос—ответ создавать проекты компонентов базы данных, которые затем могут быть «перенесены» в более производительные системы — SQL Server или Oracle.

В СУБД Microsoft Access предоставляется выбор из двух технологий (машин) баз данных (data engines): первоначальной технологии Jet и новой — Microsoft Data Engine (MSDE), совместимой с Microsoft Backoffice SQL Server (продуктом компании Microsoft для администрирования локальных корпоративных сетей).

Машина базы данных Jet хранит все данные приложения (таблицы, индексы, запросы, формы и отчеты) в одном файле базы данных с расширением .mdb, организованным с использованием ISAM (Indexed Sequential Access Method — индексно-последовательный метод доступа).

Основой MSDE является та же машина базы данных, что и в СУБД Microsoft SQL Server, предоставляющая пользователям возможность писать масштабируемые приложения на компьютере с системой Wndows 95, которые затем можно перенести в высокопроизводительные многопроцессорные кластеры (группы компьютеров), работающие под управлением системы Windows NT. Машина MSDE предоставляет также процедуру преобразования данных, позволяющую пользователям впоследствии наращивать вычислительные возможности до уровня SQL Server.

Microsoft Access, как и SQL Server, делит данные, хранящиеся в ее табличных структурах, на страницы данных размером в 2 Кбайт, что соответствует размеру стандартного кластера файла жесткого диска в операционной системе DOS. Каждая такая страница содержит одну или несколько записей. При этом запись не может занимать больше одной страницы, хотя записи типа Memo (поля примечаний) и поля объектов OLE могут храниться на отдельных страницах.

СУБД Access в качестве стандартного способа хранения записей использует запись переменной длины, а упорядочивает их с помощью индекса первичного ключа. При использовании формата хранения записи с переменной длиной каждая запись занимает только пространство, необходимое для хранения ее фактических данных.

Для создания списка связей страниц данных к каждой странице добавляется заголовок. При этом заголовок содержит два указателя: на предыдущую и следующую страницы.

Одним из преимуществ страниц данных с собственными заголовками является то, что они могут храниться в индексированном виде (в соответствии с методом доступа ISAM), т.е. в случае необходимости изменяют только указатели в заголовке страницы, а не структуру файла.

Microsoft Access обеспечивает четыре основных многопользовательских способа работы с базой данных в локальных вычислительных сетях предприятий:

- реализация файл—сервер,
- реализация клиент—сервер,
- реализация на основе репликации баз данных,
- реализация баз данных на основе Web-технологий.

Реализация файл—сервер. Базу данных Access располагают в сети таким образом, чтобы пользователи могли использовать ее совместно. В этом случае на каждой рабочей станции эксплуатируется

отдельная копия приложения и становится эффективным использование доступа к данным с помощью технологии ADO .NET

Реализация клиент—сервер. В последних версиях СУБД Access (начиная с Access 2000) предусмотрена возможность создания файлов с расширением .adp, в которых могут храниться только клиентские части приложения: формы, отчеты, макросы и модули VBA. С помощью технологии OLE DB этот файл может интегрироваться с серверной частью приложения — таблицами удаленной базы данных, хранящимися в SQL Server.

В более ранних версиях СУБД Access для достижения этого необходимо было создавать связанные таблицы, что позволяло использовать драйвер ODBC для связи с такой базой данных, как SQL Server.

Реализация на основе репликации баз данных. Репликация предполагает создание одной или нескольких копий, называемых *точными копиями* (replica) первоначальной базы данных — *проектного эталона* (design master). Проектный эталон и его точные копии называют *набором точных копий* (replica set).

При данном способе реализации любые изменения в объектах и данных передаются всем элементам набора точных копий, что происходит благодаря выполнению так называемого *процесса синхронизации.* При этом изменения в проекте объектов можно делать только в проектном эталоне, а изменения в данные можно вносить из любого элемента набора точных копий.

Реализация баз данных на основе Web-технологий. В этом случае броузер как средство навигации и просмотра отображает одну или несколько страниц доступа, которые динамически связываются с совместно используемой базой данных Access или SQL Server.

8.2. Создание серверного приложения преобразованием проекта базы данных формата Microsoft Access в формат SQL Server

Для переноса некоторых или всех объектов базы данных Microsoft Access (.mdb) в новую или существующую базу данных Microsoft SQL Server версий 2000 и 7.0 или 6.5 либо в новый проект Microsoft Access (.adp) используют мастер преобразования базы данных.

Мастер преобразования в формат Microsoft SQL Server позволяет:

- преобразовать все объекты базы данных Microsoft Access в формат *проекта Microsoft Access*, что обеспечит создание приложения типа клиент—сервер;
- преобразовать только данные или определения данных из формата базы данных Microsoft Access в формат базы данных Microsoft SQL Server;

• создать клиентскую часть базы данных в формате Microsoft Access для серверной части базы данных в формате Microsoft SQL Server, что потребует небольших изменений в приложениях, поскольку программы будут по-прежнему использовать ядро базы данных Microsoft Jet.

Перед преобразованием базы данных Microsoft Access в формат базы данных Microsoft SQL Server или проекта Microsoft Access предварительно рекомендуется выполнить следующие действия.

1. *Создать резервную копию базы данных.* Хотя мастер преобразования и не удаляет из базы данных Access данные или объекты базы данных, перед преобразованием рекомендуется создать резервную копию базы данных Microsoft Access.

2. *Убедиться, что на диске достаточно места.* На диске, где будет храниться преобразованная база данных Microsoft SQL Server, должно быть достаточно свободного места, так как мастер преобразования лучше работает, когда на диске имеется свободное пространство. Система Microsoft SQL Server автоматически увеличивает размер базы данных Microsoft SQL Server 7.0 или более поздней версии по мере ее создания.

3. *Создать уникальные индексы.* Для обновления в Microsoft Access связанная таблица должна иметь уникальный индекс. Мастер преобразования в формат Microsoft SQL Server может преобразовать существующий уникальный индекс, но не может его создать, поэтому если требуется наличие возможности обновлять таблицы, следует перед преобразованием добавить уникальные индексы в каждую таблицу Microsoft Access.

4. *Установить принтер по умолчанию.* Это требуется для использования мастера преобразования в формат Microsoft SQL Server.

5. *Присвоить необходимые разрешения на доступ к базе данных Microsoft Access.* Для выполнения преобразования всех объектов базы данных разработчик должен иметь разрешения на считывание и изменение макета.

6. *Присвоить необходимые разрешения на доступ к базе данных Microsoft SQL Server.*

Для преобразования существующей базы данных необходимы разрешения CREATE TABLE и CREATE DEFAULT.

Для построения новой базы данных необходимо разрешение CREATE DATABASE, а также разрешение SELECT на доступ к системным таблицам в главной базе данных.

Для создания новых устройств необходимо быть системным администратором.

7. *Создать при необходимости несколько дисковых устройств.* При выполнении преобразования к формату базы данных Microsoft SQL Server 6.5 перед запуском мастера преобразования может потребоваться создание новых устройств. Мастер создает все новые устройства на том же физическом диске, на котором находится глав-

ная база данных. Если на сервере установлено несколько дисков, можно поместить базу данных на одном из них, а журнал транзакций — на другом. В этом случае после сбоя диска базу данных можно восстановить.

В Microsoft SQL Server 6.5 базы данных и журналы транзакций могут распределяться по нескольким дискам, а мастер преобразования позволяет указать только один диск для базы данных и один диск для журнала транзакций. Чтобы указать для базы данных или журнала транзакций несколько устройств, следует сделать эти устройства используемыми по умолчанию.

В результате выполнения предварительных действий мастер преобразований создает отчет, содержащий подробное описание всех созданных объектов и перечень всех возникших в процессе ошибок, преобразует в формат Microsoft SQL Server и автоматически создает этот отчет в виде снимка отчета с тем же именем, что и у базы данных Microsoft Access, сохраняя его в стандартной папке базы данных.

Созданный отчет может содержать частные, конфиденциальные или просто важные сведения. Необходимо убедиться, что доступ к этому файлу запрещен потенциально опасным пользователям.

Отчет мастера преобразования в формат Microsoft SQL Server содержит:

- сведения о базе данных, включая ее размер, а также журнал транзакций и имена и размеры устройств для базы данных Microsoft SQL Server 6.5;
- параметры преобразования в формат Microsoft SQL Server, включая атрибуты таблиц, выбранных для преобразования, и способ преобразования;
- сведения о таблицах, включая сравнение значений в Microsoft Access и Microsoft SQL Server для имен, типов данных, индексов, условий на значение, значений по умолчанию, триггеров, а также режим добавления штампов времени;
- все обнаруженные ошибки, включая переполнение базы данных или журнала транзакций, недостаточные разрешения, не созданные устройства или базы данных, пропущенные таблицы, значения по умолчанию или условия на значение, не примененные условия связей, пропущенные запросы (поскольку они не могут быть преобразованы в синтаксис Microsoft SQL Server), а также ошибки преобразования элементов управления и источников записей в формах и отчетах.

Тексты баз данных Microsoft SQL Server 7.0 или более поздней версии мастер преобразует в юникод, добавив идентификатор строки во все строковые значения и префикс «n» во все типы данных.

Все типы данных Microsoft Access преобразуются в эквивалентные типы Microsoft SQL Server.

Рассмотрев технологии создания таблиц (основы серверной части приложения управления удаленными базами данных) средствами визуального проектирования СУБД Microsoft Access и преобразование базы данных формата Microsoft Access в приложение типа клиент—сервер формата Microsoft SQL Server, можно сделать следующий вывод: *особенность предлагаемых технологий сводится к широкому привлечению специалистов конкретной предметной области к разработке проекта баз данных.*

8.3. Проектирование и модификация таблиц командами SQL

В гл. 3 рассматривались операторы языка SQL. В данном подразделе рассмотрим подробнее операторы создания и модификации таблиц.

Создание таблиц баз данных. В различных СУБД организация процесса создания таблиц баз данных может быть различной в зависимости от диалектов SQL и организационной структуры конкретного предприятия, на котором создается информационная система.

Как правило, право создания таблиц обычно закрепляется за администратором базы данных (АБД).

В соответствии с стандартом ISO/EC 9075:2003 таблицы и другие объекты базы данных существуют в некоторой среде (environment). Среда состоит из одного или нескольких каталогов (catalog). В свою очередь, каталог состоит из некоторого количества схем (schema). Схема представляет собой поименованный набор объектов базы данных, которые определенным образом связаны друг с другом (таблицы с формами или запросами, запросы с формами или отчетами и т. п.). Все объекты схемы имеют определенного владельца — разработчика.

Стандарт также регламентирует механизм создания и удаления схем. Оператор создания схемы имеет следующий формат:

CREATE SHEMA [Name {имя схемы} | AUTORAZIATION Creator Identifier {имя пользователя}]

Следовательно, если создателем схемы под именем *CAPR TP* является Cidoroff, то данный оператор будет выглядеть следующим образом:

CREATE SHEMA *CAPR TP* AUTORAZIATION Cidoroff;

Схему можно удалить с помощью оператора DROP SHEMA, который имеет следующий формат:

DROP SHEMA Name [RESTRICT | CASCADE]

Ключевое слово RESTRICT (принимается по умолчанию) означает, что изначально схема должна быть пустой, иначе выполнение операции будет отменено. Если указано ключевое слово CASCADE, то при выполнении оператора будут удалены все связанные с удаляемой схемой объекты.

После создания общей структуры базы данных можно приступить к созданию таблиц с помощью оператора CREATE TABLE, который в общем виде имеет следующий формат:

```
CREATE TABLE TableName
    { (columnName dataType [Not NULL] [UNIQUE]
    [DEFAULT defaultOption] [CHECK (searchCondition)]
[,…]}
    [PRIMARY KEY (listOfColumns),]
    {    UNIQUE    (listOfColumns),] [, …]}
    {    FOREIGN KEY (listOfForeignKeyColumns)
    REFERENCES ParentTableName [(listOfCandidateKeyCo-
lumns)],
         [MATCH {PARTIAL | FULL }
         [ON UPDATE referentialAction]
         [ON UPDATE referentialAction] [, …]}
    {[CHECK (searchCondition) ] [, …]})
```

В результате выполнения оператора CREATE TABLE будет создана таблица, имя которой задается параметром *TableName*, состоящая из одного или нескольких столбцов типа *dataType*.

Для задания значения, применяемого по умолчанию при вводе данных в конкретный столбец, предусмотрена необязательная конструкция DEFAULT.

Кроме прочих значений опция определения применяемого по умолчанию значения *defaultOption* может включать в себя литералы.

Конструкция CHECK позволяет задать список значений для ввода в соответствующее поле таблицы.

Конструкция PRIMARY KEY определяет один или несколько столбцов, которые образуют первичный ключ таблицы. Если эта конструкция предусмотрена в конкретной версии SQL, то она должна применятся при создании каждой таблицы. По умолчанию для всех столбцов, представляющих собой первичный ключ, предусмотрено применение ограничения Not NULL.

При создании таблицы разрешено использование только одной конструкции PRIMARY KEY. В этом случае база данных отвергает все попытки выполнения операции INSERT или UPDATE, которые могут повлечь за собой создание строки с повторяющимся значением в столбце (столбцах), определенном конструкцией PRIMARY KEY. Таким образом, в базе данных гарантируется уникальность значений первичного ключа.

В конструкции FOREIGN KEY определяются внешний ключ (дочерней) таблицы и ее связь с другой (родительской) таблицей. Эта конструкция позволяет реализовать ограничения ссылочной целостности и состоит из следующих частей.

- Список *listOfForeignKeyColumns*, содержащий имена одного или нескольких столбцов создаваемой таблицы, которые образуют внешний ключ.
- Вспомогательная конструкция REFERENCES, указывающая на родительскую таблицу (т.е. таблицу, в которой определен соответствующий потенциальный ключ). Если список *listOfCandidateKeyColumns* опущен, предполагается, что определение внешнего ключа совпадает с определением первичного ключа родительской таблицы. В этом случае родительская таблица должна иметь в своем операторе CREATE TABLE конструкцию PRIMARY KEY.
- Необязательное правило обновления ON UPDATE для определения взаимосвязи между таблицами, указывающее, какое действие (*referentialAction*) должно выполняться при обновлении в родительской таблице потенциального ключа, соответствующего внешнему ключу дочерней таблицы. В качестве параметра *referentialAction* можно указать CASCADE, SET NULL, SET DEFAULT или NO ACTION. Если конструкция ON UPDATE опущена, то по умолчанию подразумевается, что в соответствии с значением NO ACTION никакие действия не выполняются.
- Необязательное правило удаления ON DELETE для определения взаимосвязи между таблицами, указывающее, какое действие (*referentialAction*) должно выполняться при удалении строки из родительской таблицы, которая содержит потенциальный ключ, соответствующий внешнему ключу дочерней таблицы. Определение параметра *referentialAction* совпадает с определением такого же параметра для правила ON UPDATE.
- Опция MATCH, позволяющая ввести дополнительные ограничения, касающиеся применения значений NULL в внешнем ключе. По умолчанию ограничение ссылочной целостности удовлетворяется, если любой компонент внешнего ключа имеет значение NULL или если в родительской таблице есть соответствующая строка. Если задана опция MATCH NULL, то либо все компоненты внешнего ключа должны быть пусты (NULL), либо все должны иметь непустые значения. Если задана опция MATCH PARTIAL, то либо все компоненты внешнего ключа должны быть пусты (NULL), либо в родительской таблице должна существовать хотя бы одна строка, способная удовлетворить это ограничение, если все остальные значения NULL были подставлены правильно.

В операторе создания таблицы может быть задано любое число конструкций FOREIGN KEY. Конструкция CHECK позволяет определять дополнительные ограничения. Если конструкция

CHECK используется в качестве ограничения столбца, то она может ссылаться только на определенный столбец.

Модификация таблиц. Для изменения структуры таблицы стандартом ISO/EC 9075:2003 предусмотрено применение оператора ALTER TABLE, который позволяет:

- ввести новый столбец в таблицу;
- удалить столбец из таблицы;
- ввести новое ограничение таблицы;
- удалить ограничение таблицы;
- задать для столбца значение, применяемое по умолчанию;
- удалить опцию, предусматривающую применение для столбца значения, заданного по умолчанию.

Рассмотрим основной формат оператора ALTER TABLE.

```
ALTER TABLE TableName
  [ADD [ COLUMN] ColumnName DataType [NOT NULL] [UNIQUE]
  [DEFAULT defaultOption] [CHECK (searchCondition) ]]
  [DROP [COLUMN] ColumnName [RESTRICT | CASCADE]]
  [ADD [CONSTRAINT [ConstraintName ]]
[TableConstraintDefinition]
  [DROP [CONSTRAINT [ConstraintName ]] [RESTRICT |
CASCADE]]
  [ALTER [ COLUMN] SET DEFAULT defaultOption]
  [ALTER [ COLUMN] DROP DEFAULT]
```

Как видим, многие параметры оператора ALTER TABLE совпадают с параметрами оператора GREATE TABLE.

Конструкция ADD COLUMN добавляет столбцы в базу данных.

Конструкция DROP COLUMN задает имя столбца, удаляемого из таблицы.

Ключевое слово RESTRICT указывает, что операция DROP не выполняется, если на столбец имеется ссылка в другом объекте базы данных. Это значение предусмотрено по умолчанию.

Ключевое слово CASCADE указывает, что выполнение операции DROP продолжается, и ссылки на столбец исключаются из всех объектов. Причем операция выполняется каскадно, т. е. если столбец удаляется из объекта, содержащего ссылку, то в базе данных выполняется проверка — имеются ли ссылки на удаляемый столбец в другом объекте базы данных, и такие ссылки также удаляются. Данная процедура выполняется для всех столбцов таблицы.

Удаление таблиц. Для удаления таблицы стандартом ISO/EC 9075:2003 предусмотрено применение оператора DROP TABLE, имеющего следующий формат:

```
DROP TABLE TableName [RESTRICT | CASCADE]
```

В данном формате ключевое слово RESTRICT указывает, что операция DROP не выполняется, если в базе данных имеются другие объекты, существование которых зависит от наличия удаляемой таблицы, а CASCADE означает, что выполнение операции DROP продолжается и из базы данных удаляются все зависимые объекты и объекты, зависимые от этих объектов.

8.4. Создание пользовательских представлений

В реляционных базах данных кроме объекта *таблица* (relation) существует объект *представление* (view).

Таблицы базы данных являются *базовыми отношениями*, а представления создаются из базовых отношений и являются *динамическими таблицами*, создаваемыми по требованиям отдельных пользователей в момент выполнения программы. (Фактически представление является результатом выполнения запроса на выборку, технологии создания которых будут рассмотрены далее.)

С точки зрения пользователя представление является отношением, которое существует постоянно и с которым можно работать точно так же, как с базовым отношением. Однако представление не хранится в базе данных так, как базовое отношение: в базе данных (системном каталоге базы данных) хранится лишь его определение.

Представления имеют динамический характер, т.е. любые изменения в базовых отношениях, которые могут повлиять на содержимое представления, немедленно отражаются на содержимом этого представления.

В то же время, если пользователи вносят в представление некоторые допустимые изменения, последние немедленно заносятся в базовые отношения представления.

Рассмотрим кратко назначение представлений.

- Представление обеспечивает достаточно мощный и гибкий механизм защиты, позволяющий скрыть некоторые части базы данных от определенных пользователей. Пользователь не будет иметь сведений о существовании каких-либо данных в базовых отношениях и отсутствующих в доступных только ему представлениях.
- Представление позволяет организовать одновременный многопользовательский доступ к базовым отношениям.
- Представление позволяет упрощать сложные операции с базовыми отношениями, если оно разработано на основе соединения нескольких базовых отношений.

Для создания представлений предназначен оператор GREATE VIEW, имеющий следующий формат записи:

```
GREATE VIEW ViewName [(newColumName [, …]) ]
AS subselect [WITH [CASCADED | LOCAL] CHEC OPTION]
```

Представление определяется с помощью подзапроса *subselect*, оформленного в виде оператора SELECT языка SQL. Заданный параметром *subselect* подзапрос принято называть определяющим. Если указана конструкция

```
WITH [CASCADED | LOCAL] CHEC OPTION
```

гарантируется, что в тех случаях, когда строка данных не удовлетворяет условию, указанному в конструкции WHERE определяющего запроса представления, она не будет добавлена в его базовую таблицу.

Приведем текст представления, предназначенного для выбора из базы данных «Извещение» всех полей таблицы *№ извещения* по заданному диапазону времени (начальной и конечной датам):

```
GREATE VIEW [№ извещения]
AS SELECT *
FROM [№ извещения]
WHERE ((([№ извещения].[Дата выпуска]) Between [Введите начальную дату] And [Введите конечную дату]))
```

Для удаления представлений служит оператор DROP VIEW, имеющий следующий формат записи:

```
DROP VIEW ViewName [RESTRICT | CASCADE]
```

Если в операторе задано ключевое слово RESTRICT, а в базе данных существуют объекты, зависящие от удаляемого представления, то выполнение оператора блокируется.

Если в операторе задано ключевое слово CASCADE, то при выполнении оператора будут удалены все связанные с удаляемым представлением объекты.

В случае эксплуатации СУБД на отдельном персональном компьютере использование представлений обычно имеет целью лишь упрощение структуры запросов к базе данных. Однако в многопользовательской сетевой СУБД представления играют ключевую роль в определении структуры базы данных и организации защиты информации. Основные преимущества использования представлений в данном случае заключаются в независимости от данных, обеспечении целостности информации и упрощении многотабличных запросов.

8.5. Разработка хранимых процедур

Для приложений, работающих с БД, хранимые процедуры (Stored Procedure) — это подпрограммы, которые выполняются на сервере. По отношению к БД — это объекты, которые создают-

ся и хранятся в БД и могут быть вызваны из клиентских приложений. При этом одна процедура может быть использована в любом числе клиентских приложений, что позволяет существенно экономить трудозатраты на создание прикладного программного обеспечения и эффективно применять стратегию повторного использования кода. Так же как и любые процедуры в стандартных языках программирования, хранимые процедуры могут иметь входные и выходные параметры или не иметь их вовсе.

Хранимые процедуры могут быть активизированы не только пользовательскими приложениями, но и триггерами.

Хранимые процедуры пишутся на базовом языке программирования, могут включать в себя любые операторы SQL, а также включают в себя некоторый набор операторов, управляющих ходом выполнения программ, во многом схожих с подобными операторами процедурно-ориентированных языков программирования. В коммерческих СУБД для написания текстов хранимых процедур используются собственные языки программирования. Так, в СУБД Oracle для этого используется язык PL/SQL, а в MS SQL Server — язык Transact SQL. В последних версиях Oracle для написания хранимых процедур объявлено использование языка Java.

Хранимые процедуры являются объектами БД. Каждая хранимая процедура компилируется при первом выполнении, при этом в процессе компиляции строится оптимальный план выполнения процедуры. Описание процедуры совместно с планом ее выполнения хранится в системных таблицах БД.

Для создания хранимой процедуры в SQL применяется оператор CREATE PROCEDURE.

По умолчанию выполнить хранимую процедуру могут только ее владелец, являющийся владельцем БД, и создатель. Однако владелец хранимой процедуры может делегировать права на ее запуск другим пользователям.

Имя хранимой процедуры является идентификатором в языке программирования, на котором она пишется, и должно удовлетворять всем требованиям, предъявляемым к идентификаторам в данном языке.

В MS SQL Server хранимая процедура создается оператором следующего формата:

```
CREATE PROCEDURE <имя_процедуры> [;<версия>]
{(@параметр1тип_данных}
[VARYING] [=<значение_по_умолчанию>] [OUTPUT]]
[, .параметрN.]
[ WITH
  {RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION}]
[FOR REPLICATION]
AS Тело процедуры
```

Здесь необязательное ключевое слово VARYING определяет заданное значение по умолчанию для определенного ранее параметра.

Если задано ключевое слово RECOMPILE, определяющее режим компиляции создаваемой хранимой процедуры, то данная процедура будет перекомпилироваться каждый раз, когда будет вызываться на исполнение, что может резко замедлить ее исполнение. Однако, с другой стороны, если данные, обрабатываемые данной хранимой процедурой, настолько динамичны, что предыдущий план исполнения, составленный при ее первом вызове, может быть абсолютно неэффективен при последующих вызовах, стоит применять данный параметр при создании этой процедуры.

Ключевое слово ENCRYPTION определяет режим, при котором исходный текст хранимой процедуры не сохраняется в БД. Такой режим применяется для сохранения авторского права на интеллектуальную продукцию, которой и являются хранимые процедуры.

Исходные тексты разработанных хранимых процедур не должны быть доступны администратору базы данных заказчика.

Однако все остальные параметры помимо имени хранимой процедуры являются необязательными. При этом процедуры могут быть и процедурами-функциями. Эти понятия трактуются здесь традиционно, как в языках программирования высокого уровня. Хранимая процедура-функция возвращает значение, которое присваивается переменной, определяющей имя процедуры. Явная процедура не возвращает значение, но в ней может быть использовано ключевое слово OUTPUT, определяющее, что данный параметр является выходным.

Рассмотрим несколько примеров простейших хранимых процедур.

```
/* процедура проверки наличия экземпляров конкретной книги в библиотеке. Параметры: @ISBN, шифр книги, процедура возвращает параметр, равный числу экземпляров. Если возвращается нуль, это значит, что свободных экземпляров данной книги в библиотеке нет */
CREATE PROCEDURE COUNT_EX ((@ISBN varchar (12))
AS
/* определим внутреннюю переменную */
DECLARE (@TEK_COUNT int
/* выполним соответствующий оператор SELECT, считая при этом только те экземпляры, которые в настоящий момент находятся в библиотеке */
SELECT @TEK_COUNT = select count (*)
FROM EXEMPLAR
WHERE ISBN = @ISBN
AND READER_@ID Is NULL AND EXIST = True
```

```
/* нуль означает, что ни одного свободного экземпля-
ра данной книги в библиотеке нет */
RETURN @TEK_COUNT
```

Хранимая процедура может быть вызвана несколькими способами. Простейшим способом является использование оператора EXEC:

```
EXEC <имя процедуры> <значение_входного_параметра1>...
<имя_переменной_для_выходного параметра 1>...
```

При этом все входные и выходные параметры должны быть обязательно заданы в том порядке, в котором они определены в процедуре.

Например, если надо найти число имеющихся в библиотеке экземпляров книги «Oracle8. Энциклопедия пользователя» с ISBN 966-7393-08-09, текст вызова ранее созданной хранимой процедуры может быть следующим:

```
/*Определим две переменные: @Ntek – число экземпля-
ров данной книги, имеющихся в наличии в библиотеке, и
@ISBN – международный шифр этой книги */
declare @Ntek int
DECLARE @ISBN VARCHAR(14)
/* Присвоим значение переменной @ISBN */
SELECT @ISBN = '966-7393-08-09'
/* Присвоим переменной @Ntek результаты выполнения
хранимой процедуры COUNT_EX */
EXEC @Ntek =COUNT_EX:2 @ISBN
```

Если определены несколько версий хранимой процедуры, то при вызове можно указать номер конкретной версии для исполнения. Например, во второй версии процедуры COUNT_EX последний оператор ее исполнения имеет следующий вид:

```
EXEC @Ntek =COUNT_EX:2 @ISBN
```

Однако если в процедуре определены значения входных параметров по умолчанию, то при ее запуске можно указывать значения не всех параметров. В этом случае оператор вызова процедуры можно записать в следующем виде:

```
EXEC <имя процедуры> <имя_параметра1>=<значение па-
раметра 1>... <имя_параметраN>=<значение параметраN>...
```

Например, создадим процедуру, которая считает число книг, изданных конкретным издательством в конкретном году (при создании процедуры в качестве года издания по умолчанию зададим значение текущего года):

```
CREATE PROCEDURE COUNT_BOOKS (@YEARIZD int = Year
(GetDate()).
  @PUBLICH varchar (20))
  /* процедура подсчета числа книг конкретного изда-
тельства, изданных в конкретном году. Параметры: @YEARIZD
int – год издания и @PUBLICH – название издательства */
  AS
  DECLARE @TEK_Count int
  SELECT @TEK_Count = SELEKT COUNT (ISBN)
  FROM BOOKS
  WHERE YEARIZD =@YEARIZD AND PUBLICH = @PUBLICH
  /*Одновременно с исполнением оператора SELECT ре-
зультаты его выполнения присваиваются определенной ранее
переменной @1LK_Count */
  /* при формировании результата работы процедуры сле-
дует учесть, что в библиотеке, возможно, нет ни одной
книги данного издательства заданного года. Результат
выполнения запроса SELECT в этом случае будет иметь
неопределенное значение, однако анализировать все-таки
лучше числовые значения, поэтому в качестве возвраща-
емого значения используют результаты работы специаль-
ной встроенной функции COALESCE (n1, n2, …, nm) языка
Transact SQL, которая возвращает первое конкретное зна-
чение из списка значений n1, n2, …, nm, т.е. значение,
не равное NULL */
  RETURN COALESCE (@TEK_Count, ())
```

Теперь вызовем эту процедуру, для чего подготовим переменную, в которую можно поместить результаты ее выполнения:

```
declare @N int
```

Переменная @N содержит число книг в библиотеке, изданных Издательским центром «Академия» в текущем году. Можно также обратиться к данной процедуре, задав все параметры:

```
EXEC @N = COUNT_BOOKS @PUBLICH = 'АКАДЕМИЯ' , @YEARIZD
= 2004
```

В результате получим число книг, изданных Издательским центром «Академия» в 2004 г. и имеющихся в наличии в библиотеке.

Задавая параметры по именам, необязательно использовать тот порядок, в котором они описаны при создании процедуры.

Каждая хранимая процедура является объектом БД, т.е. она имеет уникальное имя и уникальный внутренний номер в системном каталоге. При изменении текста хранимой процедуры следует сначала удалить ее как объект, хранимый в БД, и только после этого записать вместо нее новую процедуру. Отметим, что

при удалении хранимой процедуры удаляются одновременно все ее версии.

Для того чтобы автоматизировать процесс удаления старой процедуры и замены ее новой, в начале текста хранимой процедуры можно выполнить проверку наличия объекта типа «хранимая процедура» с заданным именем в системном каталоге. При наличии описания данного объекта удалить его из системного каталога. В этом случае текст хранимой процедуры предваряется специальным оператором проверки и может иметь, например, следующий вид:

```
/* проверим существование в системном каталоге объекта
с заданными именем и типом, созданного владельцем БД */
IF exists (select * from sysobjects where id =
object_id ('dbo.NEW_BOOKS') and systant & 0xf = 4)
/* если объект существует, сначала удалим его из
системного каталога*/
drop procedure dbo.NEW_BOOKS
GO
CREATE PROCEDURE NEW_BOOKS (@ISBN varchar (12) .@TITL
varchar (255),@AUTOR varchar (30).@COAUTOR varchar (30).
@ YEARIZD int. @PAGES INT,@NUM_EXEMPL INT)
/* процедура ввода новой книги с указанием числа ее
экземпляров.
Изменяемые параметры
@ISBN varchar (12)      шифр книги
@TITL varchar(255)      название
@AUTOR varchar(30)      автор
@COAUTOR varchar(30)    соавтор
@YEARIZD        int     год издания
@PAGES          int     число страниц
@NUM_EXEMPL     int     число экземпляров */
AS
/* опишем переменную, в которой будет храниться число
оставшихся неоприходованных экземпляров книги, т.е.
экземпляров, которым еще не заданы инвентарные номера */
DECLARE @TEK int
/* введем данные о книге в таблицу BOOKS */
INSERT INTO BOOKS VALUES @ISBN. @TITL. @AUTOR.
@COAUTOR.@YEARIZD@PAGES)
/* зададим значение текущего счетчика экземпляров,
оставшихся к вводу */
SELECT @TEK = @NUM_EXEMPL
/* организуем цикл для ввода новых экземпляров данной книги */
WHILE @TEK>0
/* пока число оставшихся экземпляров больше нуля */
```

```
BEGIN
/* так как для инвентарного номера экземпляра книги
задано свойство IDENTITY, его не надо вводить. СУБД
сама автоматически вычислит этот инвентарный номер,
добавив единицу к предыдущему номеру, и введет его при
выполнении оператора ввода INSERT.
Поле, определяющее присутствие экземпляра в библио-
теке (EXIST), – логическое, введем в него значение
TRUE, соответствующее наличию экземпляра книги в биб-
лиотеке*/
INSERT into EXEMPLAR ( ISBN.DATA_IN. DATA_OUT. EXIST)
VALUES (@ISBN.GetDate ( ).GetDate ( )).TRUE)
/* изменим текущее значение счетчика числа оставших-
ся экземпляров */
SELECT @TEK = @TEK - 1
END /* конец цикла ввода данных об экземпляре книги*/
GO
```

Если инкрементное поле не использовалось в качестве инвентарного номера экземпляра, можно самим назначать этот инвентарный номер, увеличив на единицу номер последнего хранимого в библиотеке экземпляра книги. Можно также попробовать просто сосчитать число существующих экземпляров в библиотеке, но если при этом некоторые из них отсутствуют, номер нового экземпляра может быть уже использован, и тогда нельзя будет ввести данные, так как система не позволит нарушить уникальность первичного ключа.

Текст процедуры в этом случае будет иметь следующий вид:

```
/* проверим существование в системном каталоге объекта
с заданными именем и типом, созданного владельцем БД */
 if exists (select * from sysobjects where id =
object_id('dbo.NEW_BOOKS') and sysstat & Oxf = 4)
/* если объект существует, сначала удалим его из
системного каталога */
drop procedure dbo. NEW_BOOKS
CREATE PROCEDURE NEW_BOOKS (@ISBN varchar (12) .@TITL
varchar (255),@AUTOR varchar (30).@COAUTOR varchar (30).@
YEARIZD int. @PAGES INT,@NUM_EXEMPL INT)
/* процедура ввода новой книги с указанием числа ее
экземпляров.
Изменяемые параметры
@ISBN varchar (12)    шифр книги
@TITL varchar(255)    название
@AUTOR varchar(30)    автор
@COAUTOR varchar(30)  соавтор
@YEARIZD   int        год издания
```

```
  @PAGES          int     число страниц
  @NUM_EXEMPL     int     число экземпляров */
  DECLARE @TEK    int
  DECLARE @INV    int
  INSERT INTO BOOKS VALUES (@ISBN.@TITL.@AUTOR.@COAUTOR.
@YEARIZD.@PAGES)
  /* зададим значение текущего счетчика экземпляров,
оставшихся к вводу */
  SELECT @TEK = @NUM_EXEMPL
  /* определим максимальное значение инвентарного но-
мера в библиотеке */
  SELECT @INV = SELECT MAX (ID_XEMPLAR)
  FROM EXEMPLAR
  /* организуем цикл для ввода новых экземпляров дан-
ной книги */
  WHILE @TEK>0
  /* пока число оставшихся экземпляров больше нуля */
  BEGIN
  INSERT INTO EXEMPLAR (ID_XEMPLAR.ISBN.DATA_IN.
DATA_OUT.EXIST)
  VALUES ((@INV.(@ISBN.GETDATE ( ). GETDATE ( ), TRUE)
  /* изменим текущие значения счетчика и инвентарного
номера */
  SELECT @TEK = @TEK - 1
  SELECT @INV = @INV + 1
  END
  /* конец цикла ввода данных об экземпляре книги */
  GO
```

Хранимые процедуры могут вызывать одна другую. Создадим хранимую процедуру, которая возвращает номер читательского билета для конкретного читателя:

```
  /* проверим существование в системном каталоге объекта
с заданными именем и типом, созданного владельцем БД */
  if exists (select * from sysobjects where id = object
fd ('dbo. CK_READER') AND sysstat & Oxf = 4)
  /* если объект существует, сначала удалим его из
системного каталога */
  drop procedure dbo.CK_READER
  /* процедура возвращает номер читательского билета,
если читатель есть, и выдает нуль в противном случае.
В качестве параметров передадим фамилию и дату рожде-
ния */
  CREATE PROCEDURE CK_READER (@FIRST_NAME varchar (30).
@BIRTH_DAY varchar (12))
  AS
```

```
/* опишем переменную, в которой будет храниться номер читательского билета*/
DECLARE @NUM_READER INT
/* определим наличие читателя */
SELECT @NUM_READER = SELECT NUM_READER
FROM READERS
WHERE FIRST_NAME = @ FIRSTJAME AND convert (varchar (8). BIRTH_DAY,4) = @BIRTH_DAY
RETURN COALESCE ((@NUM_READER.0)
```

В данной процедуре использовалась функция преобразования типа данных dataTime в тип данных varchar(8). Это было необходимо сделать для согласования типов данных при выполнении операции сравнения. Действительно, входная переменная @BIRTH_DAY имеет символьный тип (varchar), а поле базы данных BIRTH_DAY имеет тип SmallDateTime.

Хранимые процедуры допускают наличие нескольких выходных параметров, для чего каждый из этих параметров после задания ему типа данных должен иметь дополнительное ключевое слово OUTPUT. Рассмотрим пример хранимой процедуры с несколькими выходными параметрами. Создадим процедуру ввода нового читателя, причем внутри этой процедуры выполним проверку наличия в картотеке данного читателя, чтобы не назначать ему новый номер читательского билета. При этом выходными параметрами процедуры будут номер читательского билета, признак того, был ли ранее записан читатель с данными характеристиками в нашей библиотеке, а если он был записан, то сколько книг за ним числится.

Данная процедура будет иметь следующий вид:

```
/* проверим наличие данной процедуры в БД*/
if exists (select * from sysobjects where id = object_id (N'[dbo].[NEW_READER]') and OBJECTPROPERTY (id, N' Is Procedure')=1)
drop procedure [dbo].[NEW_READER]
GO
/* процедура проверки существования читателя с заданными значениями вводимых параметров возвращает новый номер читательского билета, если такого читателя ранее не было, а если он был, сообщает его старый номер и число книг, которые он должен */
CREATE PROCEDURE NEW_READER
(@NAME_READER varchar (30), (@ADRES varchar (40). @HOOM_PHONE char (9). (@WORK_PHONE char (9), @BIRTH_DAY varchar (8). @NUM_READER int OUTPUT.
/* опишем выходной параметр, определяющий номер читательского билета*/
@Y_N int OUTPUT.
```

```
/* опишем выходной параметр, определяющий, был ли
читатель ранее записан в библиотеку */
@COUNT_BOOKS int OUTPUT
/* опишем выходной параметр, определяющий число книг,
которое числится за читателем */
AS
/* опишем переменную, в которой будет храниться но-
мер читательского билета, если читатель уже был запи-
сан в библиотеку */
DECLARE @N_R int
/* определим наличие читателя */
EXEC @N_R = CK_READER @NAME_READER.@BIRTH_DAY
IF @N_R= 0 Or @N_R Is Null
/* если читатель с заданными характеристиками не
найден, т.е. переменной @N_R присвоено значение нуль
или ее значение не определено, перейдем к назначению
для нового читателя нового номера читательского биле-
та */
BEGIN
/* так как номер читательского билета определялся
как инкрементное поле, то в операторе ввода его не
указывают, система сама назначит новому читателю оче-
редной номер */
INSERT INTO RADER NAME_READER.ADRES.HOOM_PHONE,
WORK_PHONE. BIRTH_DAY)
VALUES (@NAME_READER.@ADRES.@HOOM_PHONE.
(@WORK_PHONE, Convert (smalldatetime . @BIRTH_DAY.4) )
/* в операторе INSERT следует преобразовать симво-
льную переменную @BIRTH_DAY в тип данных smalldatetime,
который определен для поля дата рождения BIRTH_DAY.
Это преобразование выполняется с помощью встроенной
функции Convert языка Transact SQL */
/* определим назначенный номер читательского билета */
SELECT @NUM_READER = NUM_READER
FROM READER
WHERE NAME_READER = @NAME_READER
Convert (varchar (8).BIRTH_DAY.4) = @BIRTH_DAY
/* снова используем функцию преобразования типа,
только в этом случае необходимо преобразовать поле
BIRTH_DAY из типа smalldatetime в тип varchar (8), в
котором задан входной параметр @BIRTH_DAY */
SELECT @Y_N =0
/* присвоим выходному параметру @Y_N значение 0 (нуль),
соответствующее тому, что данный читатель ранее в биб-
лиотеке не был записан */
SELECT @COUNT_BOOKS = 0
```

```
   /* присвоим выходному параметру, хранящему число книг,
числящихся за читателем, значение нуль */
   RETURN 1
   END
   ELSE
   /* если значение переменной @N_R не равно нулю, то
читатель с заданными характеристиками был ранее запи-
сан в библиотеке */
   BEGIN
   /* определим число книг у читателя с найденным номе-
ром читательского билета */
   SELECT @COUNT_BOOKS = COUNT(INV_NUMBER)
   FROM EXEMPLAR WHERE NUM_READER = @N_R
   SELECT @COUNT_BOOKS = COALESCE (@COUNT_BOOKS.0)
   /* присвоим выходному параметру @COUNT_BOOKS значе-
ние, равное числу книг, числящихся за читателем. Если
в предыдущем запросе @COUNT_BOOKS было присвоено не-
определенное значение, заменим его на нуль, используя
для этого встроенную функцию COALESCE(@COUNT_BOOKS.0),
которая возвращает первое определенное значение из списка
значений, заданных в качестве ее параметров */
   SELECT @Y_N = 1
   /* присвоим выходному параметру @Y_N значение 1,
соответствующее тому, что данный читатель ранее в биб-
лиотеке был записан */
```

Хранимые процедуры также играют ключевую роль в повышении быстродействия работы в сети с архитектурой клиент—сервер. В этом случае клиент обращается к серверу только для выполнения команды запуска хранимой процедуры, которая выполняется на сервере. Объем пересылаемой по сети информации при этом резко сокращается.

8.6. Разработка триггеров

Триггер — это специальный вид хранимой процедуры, которую SQL Server вызывает при выполнении операций модификации соответствующих таблиц и которая автоматически активизируется при выполнении операции, с которой он связан. При этом триггеры связываются с одной или несколькими операциями модификации над одной таблицей.

В разных коммерческих СУБД рассматриваются разные триггеры. Так, в MS SQL Server триггеры определены только как постфильтры, т.е. как триггеры, которые выполняются после свершения события.

В СУБД Oracle определены два типа триггеров: которые могут быть запущены перед реализацией операции модификации, называемые BEFORE-триггерами, и которые активизируются после выполнения соответствующей модификации аналогично триггерам MS SQL Server, называемые AFTER-триггерами.

Триггеры могут эффективно использоваться для поддержки семантической целостности БД, однако приоритет их ниже, чем приоритет правил, т.е. ограничений (constraints), задаваемых на уровне описания таблиц и связей между ними. При написании триггеров всегда надо помнить об этом. При нарушении правил целостности по связям (DRI — Declarative Referential Integrity) триггер просто может никогда не сработать.

Для создания триггеров используется специальная команда:

```
CREATE TRIGGER <имя_триггера>
ON <имя_таблицы>
FOR {[INSERT][, UPDATE] [, DELETE]}
[WITH ENCRIPTING]
AS
SQL-операторы (текст программы)
```

Имя триггера является идентификатором в встроенном языке программирования СУБД и должно удовлетворять соответствующим требованиям.

В параметре FOR задается одна или несколько операций модификации, которые и запускают триггер.

Параметр WITH ENCRIPTING имеет тот же смысл, что и для хранимых процедур, т.е. он скрывает исходный текст триггера.

Существуют следующие правила, ограничивающие состав операторов триггера:

- нельзя использовать в теле триггера операции создания объектов новой БД;
- нельзя использовать в триггере команду удаления объектов DROP для всех типов объектов БД;
- нельзя использовать в теле триггера команды изменения объектов базы данных ALTER TABLE, ALTER DATABASE;
- нельзя изменять установленные права доступа к объектам БД, т.е. выполнять команду GRAND или REVOKE;
- нельзя создать триггер для представления (VIEW);
- в отличие от хранимых процедур триггер не может возвращать никаких значений, он запускается автоматически сервером и не может связаться самостоятельно ни с одним клиентом.

Контрольные вопросы

1. Назовите основные причины, по которым Microsoft Access рекомендуется применять для разработки проекта многопользовательских удаленных баз данных.

2. Какова технология создания серверного приложения преобразованием проекта базы данных формата Microsoft Access в формат SQL Server?

3. Укажите назначение операторов в следующей структуре SQL:

```
CREATE TABLE TableName
    { (columnName dataType [Not NULL] [UNIQUE]
    [DEFAULT defaultOption] [CHECK (searchCondition) ] [,…]}
    [PRIMARY KEY (listOfColumns),]
    {      UNIQUE (listOfColumns),] [, …]}
    {      FOREIGN KEY (listOfForeignKeyColumns)
    REFERENCES ParentTableName [(listOfCandidateKeyColumns)],
           [MATCH {PARTIAL | FULL}
           [ON UPDATE referentialAction]
           [ON UPDATE referentialAction] [, …]}
    {[CHECK (searchCondition)] [, …]})
```

Каков результат выполнения данного фрагмента программы?

4. Чем отличается пользовательское представление от таблицы базы данных и как оно разрабатывается?

5. Что такое хранимая процедура и в какой части архитектуры клиент—сервер выполняется эта подпрограмма?

6. Чем отличается триггер от хранимой процедуры?

7. Назовите типы триггеров, предусмотренных в СУБД MS SQL Server и Oracle.

ЧАСТЬ IV

ПРОЕКТИРОВАНИЕ КЛИЕНТСКОЙ ЧАСТИ ПРИЛОЖЕНИЯ БАЗ ДАННЫХ

ГЛАВА 9

ОБЩИЕ ПРИНЦИПЫ ПРОЕКТИРОВАНИЯ КЛИЕНТСКОЙ ЧАСТИ БАЗ ДАННЫХ

9.1. Основные требования к разработке пользовательского интерфейса

Создание пользовательского приложения требует разработки так называемого дружественного интерфейса пользователя, т.е. организации диалога между пользователем и компьютером (клиентом и сервером).

Основным способом организации диалога является разработка диалоговых форм, которые по назначению можно подразделить на следующие группы:

- для ввода данных в таблицы;
- для ввода условий обработки информации в запросы;
- для автоматизации работы с объектами базы данных.

Формы для ввода данных в таблицы предназначаются для такой организации процедур внесения информации, которые могли бы свести к минимуму возможность ошибок оператора. Кроме того, такие формы могут служить для проведения анализа имеющихся в таблицах данных.

Формы для ввода условий обработки информации в запросы имеют назначение, аналогичное формам для ввода данных в таблицы.

Формы для автоматизации работы с объектами базы данных имеют различное назначение, например это формы-заставки, формы-меню, кнопочные формы и др.

Все эти формы и представляют собой интерфейс пользователя.

Разработка форм может производиться различными средствами визуального проектирования, например:

с помощью языков программирования (C++, Delphi, VBA);

с помощью специальных компонентов СУБД (конструкторов форм Microsoft Access, Oracle и др.).

Однако, какими бы средствами не разрабатывались формы интерфейса пользователя, необходимо учитывать следующие советы и рекомендации:

• прежде чем приступать к проектированию форм, необходимо продумать «сценарий» пользовательского интерфейса, т.е. определить последовательность появления форм на экране компьютера пользователя в соответствии с выполняемыми задачами. Фактически разработчик форм должен научиться создавать сценарии аналогично сценаристу художественных фильмов;

• каждая форма должна иметь название, которое однозначно определяет ее назначение;

• форма должна иметь привлекательный внешний вид, но при этом не должна содержать информации, не относящейся к конкретной задаче;

• формы для ввода данных в таблицы или параметров в запросы должны обеспечивать:

минимизацию возможных ошибок при вводе данных пользователем за счет согласования терминов и сокращений, ввода данных из списков и создания сообщений о допущенной ошибке;

оптимальные способы перемещения курсора (табуляцией, стрелками, указателем мыши);

получение пояснительных сообщений или инструкций при вводе данных в поля таблиц или запросов;

автоматическое закрытие формы и переход к следующей форме.

9.2. Разработка пользовательского интерфейса средствами визуального проектирования MS Access

Чтобы информационная система была удобна для работы пользователя, кроме создания эффективной модели данных (разработки состава и взаимодействия таблиц и запросов) необходимо разработать удобный дружественный пользовательский интерфейс.

Разработка интерфейса пользователя связана с настройкой панелей инструментов, созданием пользовательского меню, разработкой различных диалоговых форм.

Настройка панелей инструментов и пользовательского меню. Для создания и настройки панелей инструментов, строк меню и контекстных меню, а также для установки свойств, влияющих на их вид и работу, используется диалоговое окно *Настройка*. Для его открытия необходимо выбрать в меню *Вид* команду *Панели инструментов* и подкоманду *Настройка*.

Создание специальной панели инструментов для открытой базы данных производится в следующем порядке:

• в меню *Вид* выбрать команду *Панели инструментов*, а затем подкоманду *Настройка*;

• на вкладке *Панели инструментов* нажать кнопку [Создать];

• в поле *Панель инструментов* ввести необходимое имя и нажать кнопку [OK];

• на вкладке *Панели инструментов* нажать кнопку [Свойства];

• установить требуемые свойства и нажать кнопку [Закрыть].

Новая панель инструментов появится за диалоговым окном *Настройка*.

Чтобы закончить создание панели инструментов, следует выполнить следующие действия:

• добавить кнопки из диалогового окна *Настройка*;

• переместить или скопировать кнопку с другой панели инструментов.

Примечания: 1. К панели инструментов можно добавить меню пользователя.

2. Панели инструментов, содержащие кнопки, запускающие существующие макросы, создаются автоматически. Специальную панель инструментов можно присоединить к форме или отчету.

Создание специального контекстного меню для активной базы данных. Создание специального контекстного меню производится в следующем порядке:

• в меню *Вид* выбрать команду *Панели инструментов* и подкоманду *Настройка*;

• на вкладке *Панели инструментов* нажать кнопку [Создать];

• в поле *Панель инструментов* ввести имя и нажать кнопку [OK];

• на вкладке *Панели инструментов* нажать кнопку [Свойства];

• в поле со списком *Тип* выбрать пункт *Контекстное меню*;

• установить или снять флажок [Настройка] и нажать кнопку [Закрыть].

После выполнения указанных операций контекстное меню будет добавлено на панель инструментов.

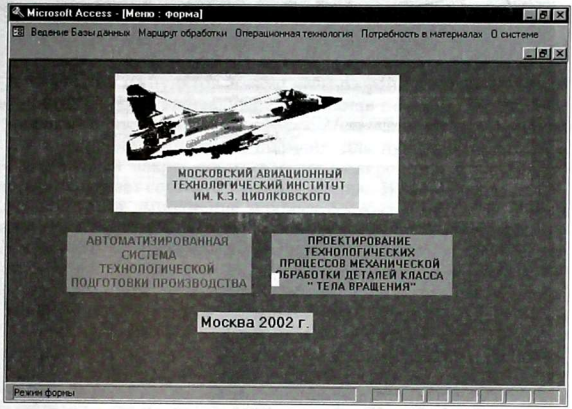
Итак, мы рассмотрели некоторые общие приемы построения систем меню при разработке пользовательского интерфейса. Теперь дадим некоторые практические рекомендации.

Очевидно, что пользовательский интерфейс представляет собой некоторую последовательность диалоговых форм. Следовательно, их созданию должна предшествовать работа по созданию «сценария» пользовательского интерфейса.

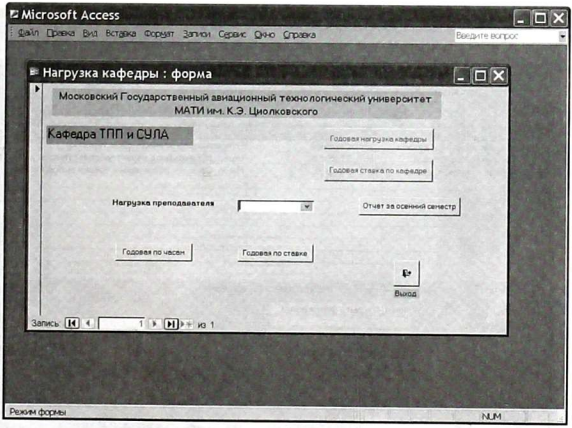
Сначала создают формы-заставки, в которых необходимо показать основное назначение разработанной системы. На этих формах также можно организовать и меню пользователя. Это могут быть ниспадающие и кнопочные меню.

Примеры форм-заставок с ниспадающим и кнопочным меню приведены на рис. 9.1.

Создание ниспадающего меню можно организовать в виде определенной последовательности макросов.



а



б

Рис. 9.1. Примеры форм-заставок с ниспадающим (*а*) и кнопочным (*б*) меню

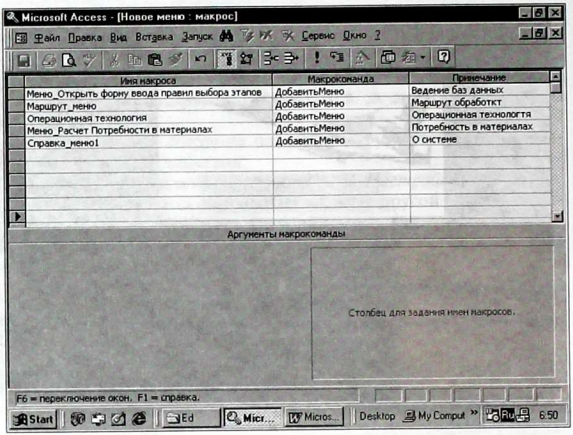


Рис. 9.2. Макрос *Новое меню* для формы-заставки, приведенной на рис. 9.1, *а*

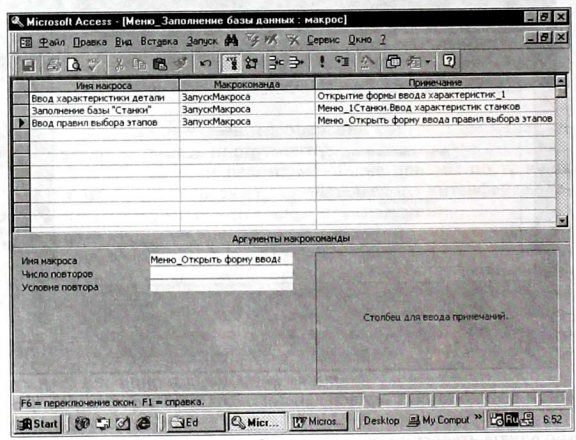


Рис. 9.3. Макрос *Меню_Заполнение базы данных*

На рис. 9.2 показан макрос *Новое меню*, заменивший традиционное меню системы Access. Макрос *Новое меню* состоит из следующих пунктов: *Ведение баз данных*, *Маршрут обработки*, *Операционная технология*, *Потребность в материалах*, *О системе*.

Для каждого из пунктов *Нового меню* разрабатывается соответствующий макрос, устанавливающий состав подменю следующего уровня. На рис. 9.3 показан макрос *Меню_Заполнение базы данных*, устанавливающий состав подменю для пункта *Ведение базы данных*. Данный макрос состоит из трех макрокоманд, каждая из которых запускает соответствующий макрос. Назначение каждого запускаемого на выполнение макроса состоит в открытии соответствующей формы ввода данных. Рассмотренным способом можно проектировать многоуровневые меню.

Кроме того, создание меню для пользователя возможно в виде командных кнопок, щелкая по которым мышью, выбирают дальнейший путь по сценарию интерфейса.

На рис. 9.4 показан пример кнопочной формы.

Из приведенных примеров видно, что на поле формы можно вставлять рисунки, которые могут ее «украсить».

На рис. 9.5 для примера показана форма с фотографией.

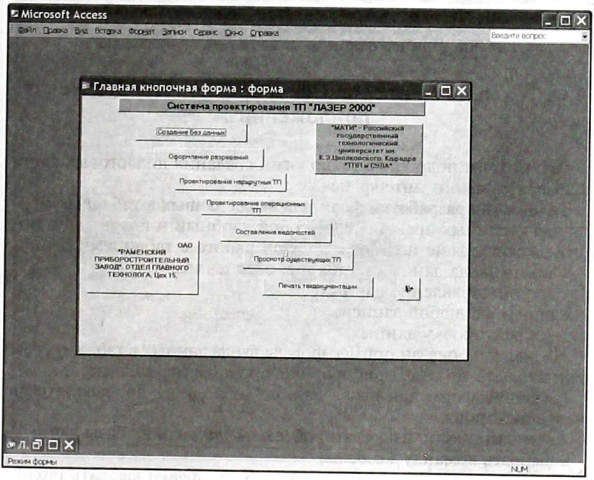


Рис. 9.4. Пример кнопочной формы

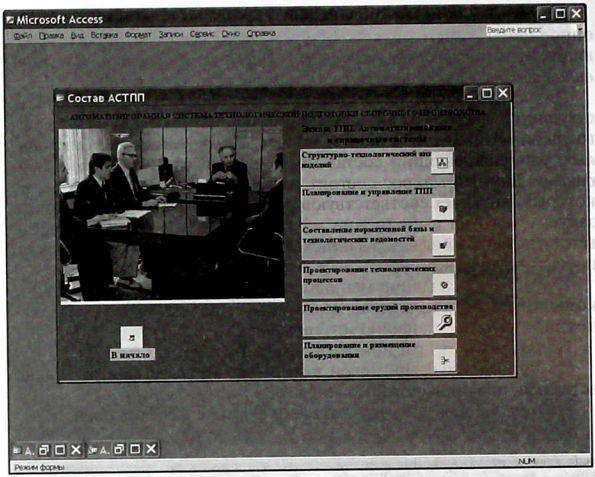


Рис. 9.5. Пример формы с вставленной фотографией

9.3. Технологии разработки форм пользовательских приложений

Рассмотрим подробнее технологии создания диалоговых форм пользовательского интерфейса.

Технология разработки форм для ввода данных в таблицы. Формы ввода данных представляют собой удобный и интуитивно понятный интерфейс пользователя при работе с таблицами.

Формы ввода данных в таблицы обеспечивают:

- ввод и добавление данных;
- просмотр любой записи;
- корректировку данных.

На рис. 9.6 показан пример формы ввода данных в таблицу базы данных учета отказов приборов в сборочном цехе завода.

Создание форм для ввода данных в таблицы производится в следующем порядке:

- выделить (активизировать) объект *Форма* в окне базы данных;
- выбрать команду *Создать*;
- в появившемся диалоговом окне *Новая форма* выбрать таблицу (из списка), для которой создается форма;

• выбрать способ создания формы.

Система Access предлагает разработчику базы данных девять способов проектирования форм (рис. 9.7):

- *Конструктор*;
- *Мастер форм*;
- *Автоформа*: *в столбец*;
- *Автоформа*: *ленточная*;
- *Автоформа*: *табличная*;
- *Автоформа*: *сводная таблица*;
- *Автоформа*: *сводная диаграмма*;
- *Диаграмма*;
- *Сводная таблица*.

Начинающим пользователям для создания форм ввода данных рекомендуется использовать способы автоматизированного проектирования *Автоформа*: *в столбец* и *Автоформа*: *ленточная*, которые можно считать самыми распространенными при создании интерфейса пользователя.

При выборе первого из указанных способов создания формы все поля таблицы будут расположены в один столбец, т. е. каждому полю будет соответствовать одна строка. При использовании

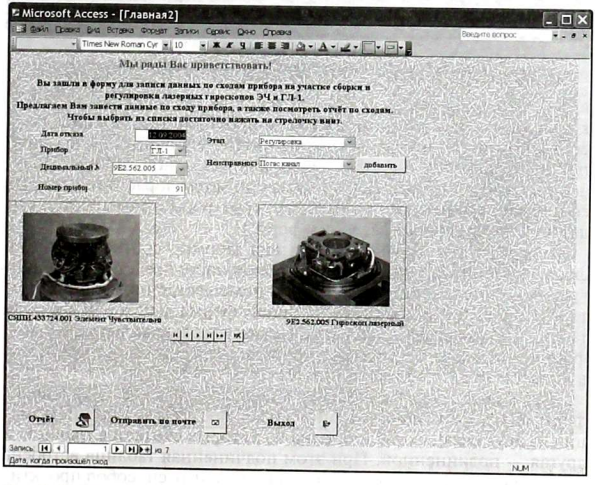


Рис. 9.6. Пример формы ввода данных в таблицу базы данных учета отказов приборов

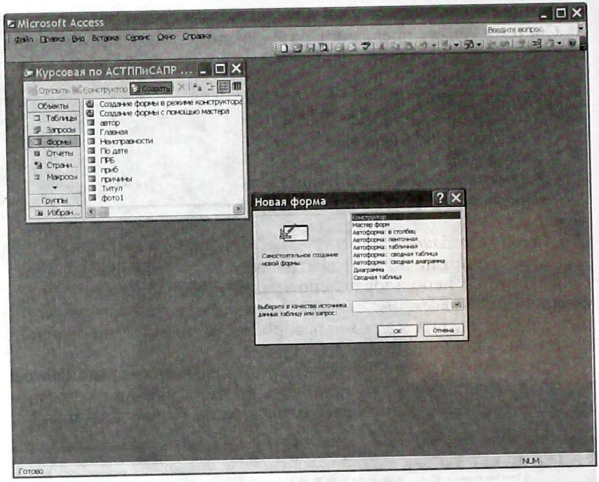
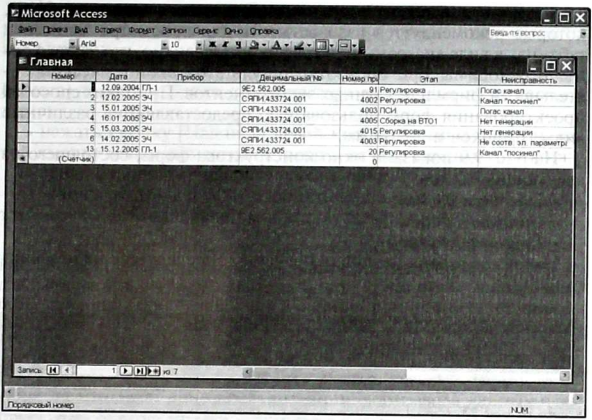


Рис. 9.7. Окно выбора способов создания форм

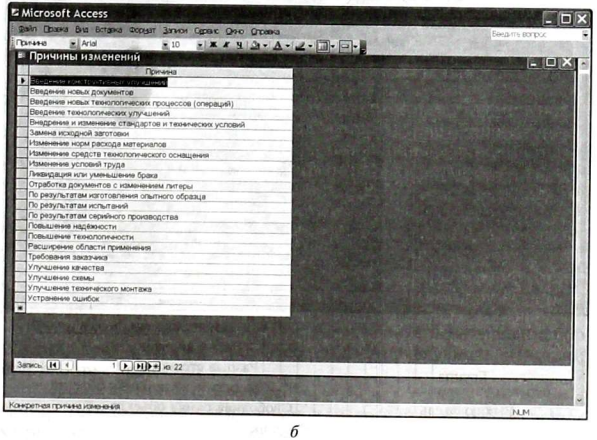
второго способа создания формы все поля таблицы автоматически располагаются в одном экранном пространстве. Подпись поля соответствует заданному в таблице имени. На одном листе (экране) располагаются поля для ввода данных одной записи. Полученную этими способами форму можно затем доработать, открыв ее в режиме *Конструктор*. (Конструктор форм применяют также для самостоятельного создания форм.)

Автоформа: *табличная* — это вид формы, соответствующий форме таблицы. В такой форме на одном листе представлено одновременно столько записей, сколько помещается на экране монитора. На рис. 9.8, *а* показана табличная форма, в которой все поля таблицы не помещаются на одной строке экрана, а на рис. 9.8, *б* — форма ввода данных в таблицу, состоящая из одного столбца, т. е. в которой каждая запись занимает одну строку.

Автоформа: *сводная таблица* и *Сводная таблица* — это виды форм, в которых одновременно может представляться информация из двух связных таблиц, одна из которых считается главной, а другая — подчиненной. При этом подчиненная таблица встроена в форму главной таблицы. При первом из этих способов проектирования таблица формируется автоматически, а при втором — с минимальными затратами разработчика.



а



б

Рис. 9.8. Табличные формы с несколькими полями (*а*) и с одним полем (*б*)

Автоформа: *Сводная диаграмма* и *Диаграмма* — это виды форм, которые рекомендуется разрабатывать для просмотра записей в таблицах в виде диаграмм или графиков. Очевидно, что такие формы необходимы при обработке результатов экономической деятельности фирм или научных экспериментов. При таких способах проектирования форм пользователю предоставляются различные виды графиков и диаграмм.

На рис. 9.9 показано окно конструктора форм, состоящего из следующих блоков:

- *Заголовок формы*;
- *Область данных*;
- *Примечание формы*.

Назначение этих блоков однозначно определяется их названиями.

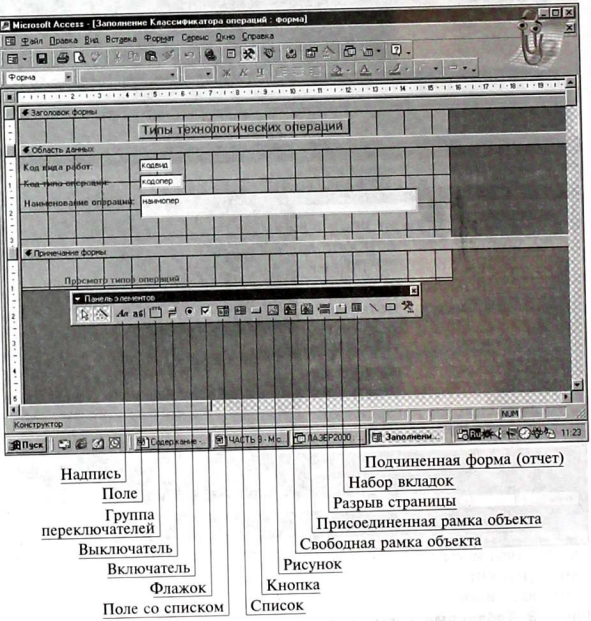


Рис. 9.9. Окно конструктора форм

На панели элементов конструктора форм расположены кнопки команд управления. Рассмотрим кратко назначение каждой из этих кнопок.

Надпись — команда, предназначенная для ввода надписей (текста) в любом блоке конструктора форм, для чего необходимо:

- нажать кнопку [Надпись] левой кнопкой мыши и при нажатой кнопке мыши поместить курсор в начало вводимого текста;
- отпустить кнопку мыши и ввести текст.

Технология ввода и оформления текста полностью аналогична технологии работы с текстом в редакторе Word.

Примечание. Чтобы разместить текст надписи в нескольких строках, в конце первой строки следует нажать клавиши [CTRL] + [ENTER] для ввода символа возврата каретки. В этом случае по мере ввода текст будет автоматически переноситься в последующие строки, а максимальная ширина надписи определится длиной первой его строки.

Поле, *Поле со списком*, *Список* — команды, предназначенные для создания соответствующих полей ввода данных. Их использование необходимо при проектировании форм ввода данных в запросы, технологии создания которых будут рассмотрены далее.

Если форма ввода данных в таблицу разрабатывалась одним из автоматических методов, ее поля автоматически размещаются в области данных конструктора форм и пользователю не надо обращаться к указанным командам.

При самостоятельном конструировании формы ввода данных в таблицу размещение полей производится «перетаскиванием» их из списка полей таблицы. Для этого в режиме конструктора необходимо активизировать команду меню *Вид*, выбрать команду *Список полей* и из появившегося при этом списка последовательно «перетащить» поля таблицы, размещая их в области данных конструктора таблиц (рис. 9.10). Последовательность переноса и размещения полей должна соответствовать предполагаемой последовательности ввода данных.

Группа переключателей, *Выключатель*, *Включатель*, *Флажок* — элементы, предназначенные для организации ввода данных в поля логического типа.

Кнопка — элемент управления, содержащий некоторый набор команд, предназначенных для создания управляющих действий при работе с формами.

Рисунок — команда, предназначенная для вставки рисунков в форму. Технология вставки рисунков аналогична технологии вставке рисунков в документ Word.

Свободная рамка объекта — окно, в котором отображаются данные полей таблицы типа OLE-объектов. При автоматическом создании таблиц окна в форме создаются также автоматически.

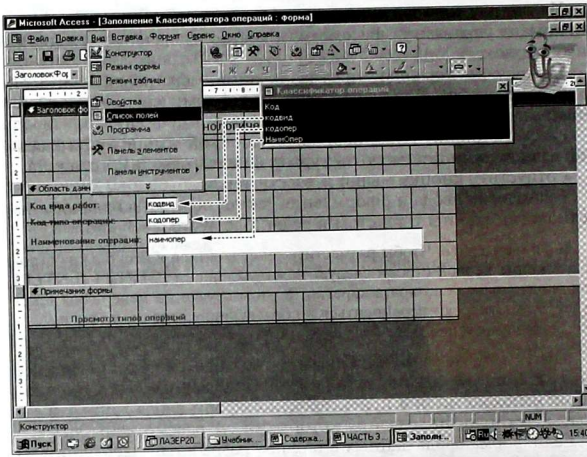


Рис. 9.10. Окно конструктора при самостоятельном проектировании формы

Присоединенная рамка объекта — окно, в котором можно разместить OLE-объект, находящийся в другом файле или другой БД.

Разрыв страницы — команда, применяемая в случае если поля для ввода данных не помещаются на одной странице (экране дисплея).

Набор вкладок — команда, применяемая в случае если поля для ввода данных не помещаются на одной странице (экране дисплея). При использовании набора вкладок рекомендуется производить группирование полей по каким-либо признакам и для каждой группы создать соответствующую вкладку. Технология размещения полей на вкладке в процессе конструирования формы основана на «перетаскивании» поля из списка.

На рис. 9.11 показана форма ввода данных с открытой вкладкой *Переходы*.

Подчиненная форма (*отчет*) — команда, применяемая при разработке составных форм, которые, как правило, разрабатываются для таблиц, связанных отношениями «один ко многим». В этом случае одна таблица является главной, а другая — подчиненной.

Аналогично при проектировании составных форм ввода данных одну из форм также будем называть главной, а другую — подчиненной.

Схему разработки составных форм можно представить в следующем виде:

- разработать форму ввода данных в подчиненную таблицу;
- разработать форму ввода данных в главную таблицу, предусмотрев в ней область для вставки подчиненной формы и используя для этого команду (кнопку) *Подчиненная форма*.

На рис. 9.12 показан пример конструирования составной формы ввода информации при создании баз данных для автоматизированного проектирования маршрутных карт технологических процессов сборки в САПР «ЛАЗЕР 2000» в режиме конструктора. Главная форма состоит из двух частей, предназначенных для ввода данных в главную и подчиненную таблицы.

Технология разработки форм для ввода данных в запросы. Рассмотрим технологию создания параметрических запросов и связанных с ними форм для ввода условий отбора данных.

Разработку специальных форм для ввода условий отбора данных в запросы обусловливают следующие факторы:

- необходимость разработки пользовательских представлений при разработке сетевых баз данных, организованных по архитектуре клиент—сервер;

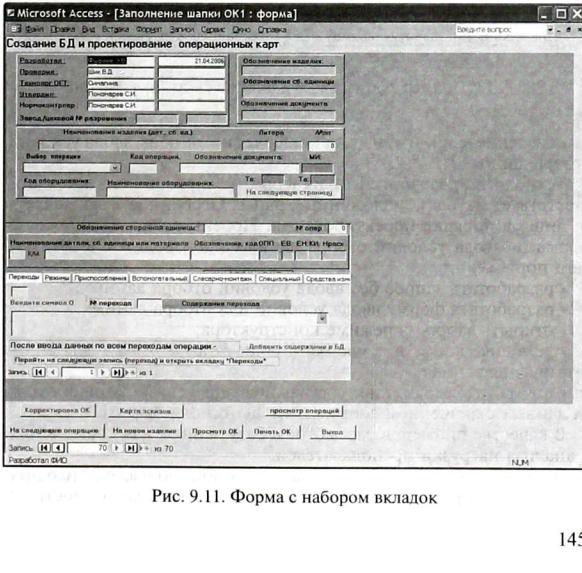
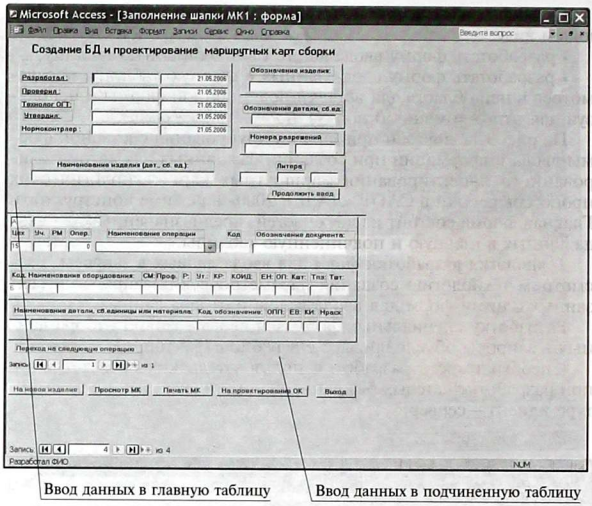


Рис. 9.11. Форма с набором вкладок



Ввод данных в главную таблицу

Ввод данных в подчиненную таблицу

Рис. 9.12. Пример формы ввода данных в связанные таблицы

• сведение к минимуму или исключение ошибок ввода условий отбора данных.

Технология разработки таких форм и запросов полностью соответствует методам, изложенным ранее. Особенность заключается лишь в порядке проектирования. Проектирование запросов с формами ввода условий отбора данных производится в следующем порядке:

• разработать запрос без ввода условий отбора данных;
• разработать форму ввода условий для отбора данных;
• открыть запрос в режиме конструктора;
• установить курсор в ячейку строки *Условие отбора* для соответствующего поля;
• построить выражение, устанавливающее связь условия отбора данных с значением соответствующего поля формы.

В качестве примеров рассмотрим базу данных для составления и анализа нагрузки преподавателей.

В процессе работы с базой данных достаточно часто приходится делать выборку записей из таблицы *Нагрузка* в зависимости от

фамилии преподавателя. Для такой выборки был разработан соответствующий запрос (рис. 9.13).

Для выполнения запроса в ячейку поля *ФИО* в строке *Условие отбора* вводят фамилию, имя и отчество преподавателя.

Для ввода фамилии была разработана форма, показанная на рис. 9.14. В этой форме предусмотрено специальное поле со списком для ввода фамилий преподавателей.

После разработки формы в соответствующем запросе *Нагрузка преподавателя*: *запрос на выборку* (см. рис. 9.13) в строке *Условие отбора*, в ячейке поля *ФИО* построено выражение следующего вида:

```
[Forms]![Нагрузка кафедры]![ПолеСоСписком0]
```

где Forms — ключевое слово, обозначающее объект БД (в данном случае — форму);

Нагрузка кафедры — имя объекта базы данных (формы);

ПолеСоСписком0 — имя поля в форме, значения которого являются условиями отбора данных для поля *ФИО* в запросе *Нагрузка преподавателя*;

скобки [ ] и знак ! — элементы «грамматики» построения выражений.

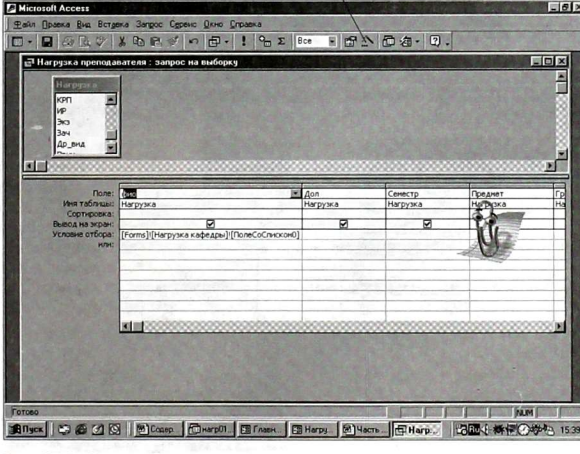


Рис. 9.13. Окно конструктора для создания запроса на выбор данных по условию

Построение выражений проще всего выполнять с применением мастера *Построитель выражений* (рис. 9.15), для чего после разработки формы следует:

- открыть запрос в режиме конструктора;
- установить курсор в ячейку строки *Условия отбора*;
- открыть окно *Построитель выражений*;
- построить требуемое выражение.

После открытия окна *Построитель выражений* необходимо выполнить следующие действия:

- выбрать объект БД, щелкнув мышью по соответствующему значку (в данном примере — Forms), после чего откроется список всех объектов данного типа;
- выбрать из списка имя объекта (формы). В результате выбора в окне элементов объекта БД появится список всех элементов (полей, подписей, кнопок и др.);
- выбрать элемент объекта БД (в данном примере — *ПолеСоСписком0*).

В результате этих действий сформируется выражение, связывающее условие отбора данных в запросе с вводимыми значениями в поле формы.

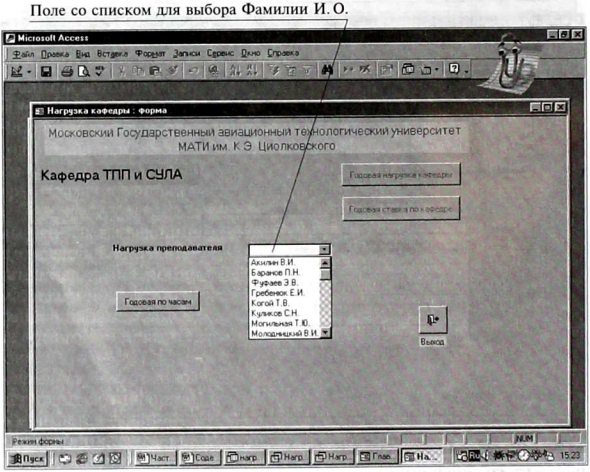


Рис. 9.14. Форма ввода условий выборки в запрос

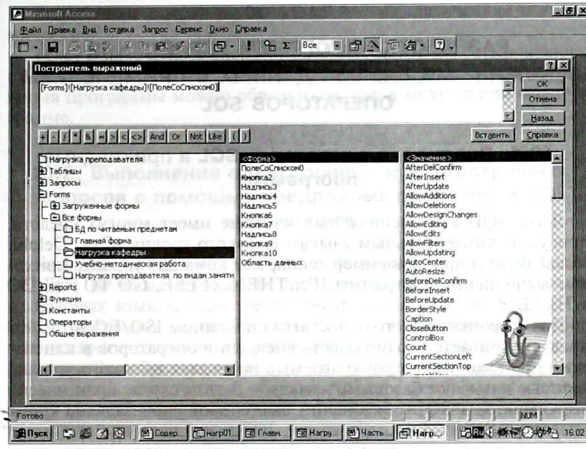


Рис. 9.15. Окно мастера *Построитель выражений*

Контрольные вопросы

1. Для чего служат кнопочные формы и формы-заставки?

2. Решение каких задач работы пользователя должны обеспечивать формы ввода данных в таблицы?

3. Дайте характеристики следующих способов проектирования форм, которые предлагает разработчику базы данных СУБД Access: *Конструктор*; *Мастер форм*; *Автоформа*: *в столбец*; *Автоформа*: *ленточная*; *Автоформа*: *табличная*; *Автоформа*: *сводная таблица*; *Автоформа*: *сводная диаграмма*; *Диаграмма*; *Сводная таблица*.

4. Какова последовательность проектирования запросов с формами ввода условий отбора данных?

5. Что представляет собой элемент приложения *Отчет*, для каких целей он разрабатывается и чем отличается от элемента *Форма*?

ГЛАВА 10

РАЗРАБОТКА ПРОГРАММ УПРАВЛЕНИЯ УДАЛЕННЫМИ БАЗАМИ ДАННЫХ С ПРИМЕНЕНИЕМ ОПЕРАТОРОВ SQL

10.1. Внедрение операторов SQL в прикладные программы

Язык SQL как специальный язык не имеет многих средств, присущих универсальным языкам высокого уровня (C++, Delphi, Visual Basic и др.), например таких, как команды управления ходом выполнения программы IF...THEN...ELSE, GO TO или DO WHELE.

Для устранения этого недостатка стандартом ISO/EC 9075:2003 предусматривается возможность внедрения операторов в клиентские приложения, разрабатываемые на языках высокого уровня, которые называют базовыми языками. В этом случае практически все операторы SQL однозначно выполняют свои задачи, кроме оператора SELECT.

На практике существует два способа использования операторов SQL в программах базового языка: внедрение операторов и применение программного интерфейса.

Внедрение операторов SQL. При использовании данного способа отдельные операторы SQL внедряются непосредственно в текст исходной программы и чередуются с операторами базового языка, что позволяет создавать приложения, которые могут непосредственно обращаться к таблицам базы данных и работать с ними. В этом случае специальные программы — предкомпиляторы — преобразуют текст исходной программы, производя замену операторов SQL соответствующими вызовами процедур СУБД, после чего этот текст компилируется. Стандарт ISO/EC 9075:2003 предусматривает обязательную поддержку внедренных операторов SQL для языков программирования ADA, C, COBOL, Fortran, Pascal, PL/1.

Применение программного интерфейса — API. Данный способ использования операторов SQL состоит в том, что программисту предоставляется возможность с помощью API обращаться к стандартному набору функций из создаваемых им программ.

Отличие данного способа от предыдущего состоит в том, что устраняется необходимость предкомпиляции текста программы, т.е. в этом случае текст исходной программы более удобен для анализа.

Соответствующие API имеются в СУБД Oracle.

В СУБД Access предлагается программный интерфейс ADO, представляющий собой надстройку над интерфейсом ODBC.

Внедренные операторы подразделяют на статические и динамические.

Текст *статического* оператора должен быть полностью внедрен в текст программы, а к *динамическому* оператору во время выполнения программы можно обращаться, как к модулю или подпрограмме.

10.2. Выполнение однострочных и многострочных запросов с помощью внедренных операторов SQL и курсоров

Как уже говорилось, внедрение всех операторов SQL кроме SELECT не вызывает особых проблем при разработке программ на базовых языках. Простое же внедрение оператора SELECT в текст программы приводит к тому, что результирующий набор данных *будет состоять только из одной строки соответствующей таблицы*. Это обусловлено тем, что операторы языков программирования высокого уровня работают только с отдельными элементами, представляющими собой отдельные строки данных, в то время как оператор языка SQL может обрабатывать произвольное число строк данных. Для устранения такой несогласованности в языке SQL предусмотрены специальные функции, позволяющие связывать переменные базового языка со строками таблиц баз данных, которые обрабатываются (возвращаются) по одной при каждом последовательном обращении. С учетом этого все запросы на языке SQL подразделяют на две группы: *однострочные* и *многострочные*. Очевидно, что результатом выполнения однострочного запроса является только одна строка данных. Результатом же выполнения многострочного запроса могут быть структуры, состоящие из произвольного числа строк от нуля до *n*.

Однострочные запросы. Язык SQL для реализации внедренных однострочных запросов предусматривает использование специального *оператора единичной выборки* с форматом, не отличающимся от формата оператора SELECT, за исключением дополнительной конструкции INTO, предназначенной для указания имен переменных базового языка, в которые следует поместить результаты запроса. Конструкция INTO должна следовать непосредственно за списком полей оператора SELECT. Между выражениями в списке SELECT и переменными базового языка в конструкции INTO должно существовать взаимно-однозначное соответствие для обеспечения выборки сведений из таблицы базы данных.

Рассмотрим пример. Пусть необходимо выбрать сведения о владельце недвижимости с номером CO21 из таблицы с именем Privat-Ower.

Используем следующую конструкцию оператора SELECT:

```
EXEC SQL SELECT f Name, lName, address
INTO : f irstName, :lastName, :address :adressInd
FROM PrivatOwer
WHERE ownerNo = 'CO21'
```

Здесь значение столбца f Name будет помещено в переменную базового языка f irstName, значение столбца lName — в переменную lastName, а значение столбца address — в переменную address (вместе с индикатором значения NULL, для которого используется переменная adressInd).

Естественно, все применяемые в данном операторе переменные должны быть объявлены заранее согласно правилам базового языка.

Многострочные запросы. Для реализации запросов, в результате выполнения которых может быть получено произвольное количество строк, язык SQL предоставляет программисту механизмы выборки данных, основанные на использовании курсоров.

Например, в языке PL/SQL курсор позволяет программе построчно обрабатывать результаты выполнения запроса, т.е. в этом случае он представляет собой указатель на определенную строку в результирующем наборе данных. Курсор можно передвигать с одной строки на другую и после обработки одной строки переходить к следующей.

Перед работой курсор должен быть объявлен и открыт, а после завершения работы — закрыт.

Для объявления курсора используется следующая структура:

```
EXEC SQL DECLARE <имя курсора> CURSOR FOR <запрос>
```

Рассмотрим пример текста программы, объявляющей курсор:

```
EXEC SQL DECLARE propertyCursor CURSOR FOR
SELECT propertyNo, street, city
FROM PropertyForRent
WHERE staffNo = 'SL41'
```

В данном примере объявляется курсор с именем propertyCursor для выбора данных из таблицы PropertyForRent по полям propertyNo, street, city при условии, что значение переменной staff No равно 'SL41'

Для открытия объявленного курсора применяется оператор OPEN, имеющий следующую структуру:

```
EXEC SQL OPEN propertyCursor FOR READONLY
```

После открытия курсора производится выбор информации из таблицы с применением оператора FETCH согласно заданным в курсоре условиям:

```
EXEC SQL FETCH propertyCursor
INTO : propertyNo, street, city;
```

При обработке оператора FETCH значение столбца propertyNo будет помещено в переменную базового языка propertyNo, а значение столбцов street и city — соответственно в переменные street и city.

Поскольку при выполнении запроса оператор FETCH обрабатывает только одну строку таблицы, в тексте программы на базовом языке он должен быть помещен внутрь соответствующего цикла, обеспечивающего требуемое число просмотров всех записей таблицы. В этом случае СУБД помещает в переменную SQLCODE значение NOT FOUND.

Если таблица не содержит ни одной строки данных, при выполнении запроса в первом же шаге цикла будет обнаружено их отсутствие и переменной SQLCODE вернется значение NOT FOUND.

После выполнения запроса курсор должен быть закрыт с помощью оператора CLOSE, имеющего следующую структуру:

```
EXEC SQL CLOSE propertyCursor;
```

10.3. Модификация таблиц баз данных с помощью курсоров

Курсор может быть предназначен как для чтения, так и для обновления таблиц баз данных. Обновлять данные в таблицах (обновляемый курсор) курсор может с использованием оператора UPDATE или DELETE CURRENT.

Если курсор является обновляемым, то согласно стандарту ISO/EC 9075—2003 конкретные разработчики СУБД могут вносить в него свои функции.

Так, в СУБД Oracle для создания обновляемых курсоров используют следующие дополнения к оператору DECLARE CURSOR:

```
EXEC SQL DECLARE < имя курсора > CURSOR FOR < имя
таблицы > FOR UPDATE OF < имена всех столбцов таблицы >
```

Более того, эти же столбцы должны быть перечислены и в списке конструкции SELECT. Вид операторов обновления данных можно показать на следующем примере:

```
EXEC SQL UPDATE TableName
SET columName = dataValue [,…]
WHERE CURRENT OF cursorName
```

В этом примере параметр cursorName представляет собой имя открытого обновляемого курсора. Конструкция WHERE исполь-

зуется для определения строки, на которую в данный момент указывает курсор, и все вносимые обновления будут выполняться только для этой строки. Каждое имя столбца обновляемой таблицы в конструкции SET должно быть указано в операторе DECLARE CURSOR.

Удаление строк можно выполнять с помощью обновляемых курсоров, например имеющих следующую структуру:

```
EXEC SQL DELETE FROM PropertyForRent
WHERE CURRENT OF propertyCursor
```

Выполнение такого курсора приведет к удалению строки таблицы, связанной с текущим положением курсора propertyCursor.

Для того чтобы можно было применять позиционные операторы удаления (DELETE) и модификации (UPDATE), курсор должен удовлетворять следующим требованиям:

- запрос, связанный с курсором, должен считывать данные из одной исходной таблицы, т.е. в предложении FROM запроса SELECT, связанного с определением курсора (DECLARE CURSOR), должна быть задана только одна таблица;
- в запросе не может присутствовать параметр упорядочения ORDER BY. Для того чтобы сохранялось взаимно-однозначное соответствие строк курсора и исходной таблицы, курсор не должен идентифицировать упорядоченный набор данных;
- в запросе не должно присутствовать ключевое слово DISTINCT;
- запрос не должен содержать операций группировки, т.е. в нем не должно присутствовать предложение GROUP BY или HAVING;
- пользователь, который хочет применить операции позиционного удаления или обновления, должен иметь соответствующие права на выполнение данных операций над базовой таблицей.

Использование курсора для операций обновления значительно усложняет работу СУБД, поэтому операции, связанные с позиционной модификацией, выполняются гораздо медленнее, чем операции с курсорами, которые используются только для чтения. Именно поэтому рекомендуется обязательно указывать в операторе определения курсора предложение READ ONLY, если вы не собираетесь использовать данный курсор для операций модификации. По умолчанию, если нет дополнительных указаний, СУБД создает курсор с возможностью модификации.

Курсоры — удобное средство для формирования бизнес-логики приложений, однако следует помнить, что если вы открываете курсор с возможностью модификации, СУБД блокирует все строки базовой таблицы, вошедшие в него, и тем самым блокируется работа с данной таблицей других пользователей.

Чтобы свести к минимуму число требуемых блокировок, при работе интерактивных программ следует придерживаться следующих правил:

- делать транзакции как можно короче;
- выполнять оператор завершения COMMIT после каждого запроса и как можно скорее после изменений, сделанных программой;
- избегать программ, в которых осуществляется интенсивное взаимодействие с пользователем или просмотр очень большого числа строк данных;
- по возможности не применять прокручиваемые курсоры (SCROLL), так как они требуют блокирования всех строк выборки, связанных с открытым курсором;
- использовать простой последовательный курсор, который позволит системе разблокировать текущую строку, как только будет выполнена операция FETCH, что минимизирует блокировки других пользователей, работающих параллельно с вами и использующих те же таблицы;
- по возможности определять курсор как READ ONLY;
- так как при рассмотрении модели клиент—сервер определено, что в развитых моделях серверов баз данных большая часть бизнес-логики клиентского приложения выполняется именно на сервере, следует использовать специальные объекты, называемые хранимыми процедурами и хранящиеся в БД, как таблицы и другие базовые объекты.

Курсоры, которые могут быть использованы в приложениях, обычно подразделяются на курсоры сервера и курсоры клиента.

Курсоры сервера создаются и выполняются на сервере, и данные, связанные с ними, не пересылаются на компьютер клиента. Курсоры сервера определяются обычно в хранимых процедурах или триггерах.

Курсоры клиента определяются в прикладных программах, выполняемых на компьютере клиента. Набор строк, связанный с таким курсором, пересылается на компьютер клиента и там обрабатывается. Если с курсором связан большой набор данных, то операция пересылки набора строк, связанных с курсором, может занять значительное время и значительные ресурсы сети и клиентского компьютера.

Конечно, курсоры сервера более экономичны и выполняются быстрее, а следовательно, рекомендуется трансформировать логику работы вашего приложения таким образом, чтобы как можно чаще вместо курсоров клиента использовать курсоры сервера.

Контрольные вопросы

1. В каких двух режимах может осуществляться доступ к БД средствами языка SQL?

2. Какие операторы, присущие универсальным языкам программирования, отсутствуют в языке SQL?

3. На какие этапы можно условно подразделить процесс выполнения запросов операторами SQL?

4. Какие два типа запросов различают во встроенном SQL?

5. Каково назначение оператора INTO?

6. Что представляет собой курсор?

7. Для чего используются курсоры в прикладных программах?

8. Что означают операторы DECLARE CURSOR, OPEN, FETCH, CLOSE?

9. Что представляет собой хранимая процедура?

10. Какие языки программирования используют в коммерческих СУБД для написания текстов хранимых процедур?

11. В чем состоит отличие триггера от хранимой процедуры?

ГЛАВА 11

WEB-ТЕХНОЛОГИИ В РАЗРАБОТКЕ УДАЛЕННЫХ БАЗ ДАННЫХ

11.1. Введение в Интернет и среду WWW

Интернет представляет собой совокупность взаимосвязанных компьютерных сетей мирового масштаба, т.е. Интернет состоит из множества отдельных, но связанных между собой сетей, принадлежащих коммерческим, образовательным и правительственным организациям, а также поставщикам услуг, или так называемым провайдерам Интернета (Internet Service Provider — ISP).

Сегодня широко используются такие услуги Интернета, как электронная почта (e-mail), средства проведения конференций и бесед, средства удаленного доступа к компьютерам, средства передачи и приема файлов.

Основы этой сети были заложены в конце 1960-х — начале 1970-х годов при выполнении экспериментального проекта ARPANET (Advanced Research Projects Agency Network) Министерства обороны США, целью которого было исследование возможности создания сетей, сохраняющих работоспособность при частичных повреждениях (например, при взрывах ядерных бомб).

В 1982 г. в качестве стандартных протоколов связи для сети ARPANET приняты протоколы TCP/IP (Transmission Control Protocol/Internet Protocol). Протокол TCP обеспечивает бесперебойную доставку сообщений с одного компьютера на другой, а протокол IP управляет передачей и приемом пакетов данных между разными компьютерами на основе четырехбайтового адреса назначения (IP-адреса), который присваивается той или иной организации уполномоченными представителями Интернета.

Термин TCP/IP иногда применяют к целому семейству протоколов сети Интернет, работающих на основе протоколов TCP/IP:

- FTP (File Transfer Protocol) — протокол передачи файлов;
- SMTP (Simple Mail Transfer Protocol) — простой протокол электронной почты;
- Telnet (Telecommunication network) — телекоммуникационная сеть;
- DNS (Domain Name Service) — служба доменных имен;
- POP (Post Office Protocol) — почтовый протокол и др.

Сеть Интернет имеет еще одно популярное название (особенно в средствах массовой информации) — информационная супермагистраль (Information Superhighway). Это ее метафорическое

название как будущей всемирной сети, которая обеспечит связь, доступ к информации и оперативным службам пользователям во всем мире.

World Wide Web (WWW), или коротко среда Web, — это гипермедийная система, предоставляющая возможность просматривать любую информацию в сети Интернет с помощью механизма гиперссылок.

Информация в среде WWW размещается на Web-страницах, оформленных в виде текста, графики, аудио- и видеоматериалов. Дополнительно Web-страница может содержать гиперссылки на другие Web-страницы.

Стандарты обмена электронной почтой и публикации Web-страниц используются не только в глобальной сети Интернет, но и в закрытых корпоративных сетях, которые и принято называть внутренними.

Внутренняя сеть — это Web-узел или группа узлов, принадлежащих одной организации и доступных только ее членам.

Закрытая внутренняя сеть подсоединяется к Интернету с помощью *брандмауэра*, позволяющего регламентировать состав передаваемой и получаемой информации.

Например, отдельным сотрудникам организации может быть разрешено использование только внешней электронной почты и разрешен доступ к внешним Web-узлам, а внешним пользователям разрешено только отправлять почту адресатам внутри этой организации и запрещено просматривать содержимое Web-страниц, опубликованных в пределах внутренней сети.

Защищенные внутренние сети являются наиболее быстро растущим сегментом Интернета, так как они гораздо дешевле и проще в управлении, чем закрытее частные сети на основе специализированных протоколов.

Внешняя сеть — это внутренняя сеть, которая может быть доступна внешним пользователям.

По сравнению с внутренней сетью, которая находится за брандмауэром и доступна только членам конкретной организации, внешняя сеть обеспечивает различные уровни доступа и для внешних пользователей.

Доступ к внешней сети обычно возможен только при условии наличия у пользователя прав доступа (учетного имени и пароля) к тем или иным ресурсам внешней сети.

В настоящее время внешние сети становятся популярным средством обмена данными между деловыми партнерами, что является обязательным в условиях внедрения на предприятиях принципов управления на основе CALS-технологий.

Применение для общения между деловыми партнерами Интернет-технологий технически осуществляется достаточно просто и с минимальными затратами.

Успех технологии Web обусловлен в основном ее простотой, т.е. она позволяет без особых хлопот предоставлять, использовать и ссылаться на информацию, территориально распределенную по всему земному шару.

Технология Web совместима также с другими существующими коммуникационными протоколами: Gopher, FTP, NNTP (Network News Transfer Protocol) и Telnet (для сеансов удаленного входа в систему).

Среда Web состоит из сети компьютеров, которые могут действовать либо как серверы, предоставляющие информацию, либо как клиенты, запрашивающие информацию, и которые обычно называют броузерами.

Примерами Web-серверов являются такие программные пакеты, как HTTP-сервер Apache, сервер IIS (Internet Information Server) компании Microsoft, Enterprise Server, WebLogic Server и др.

Примерами броузеров являются программы Microsoft Internet Explorer и Netscape Navigator.

Основная часть информации в среде Web хранится в документах, описанных на языке HTML (HyperText Markup Language) — языке гипертекстовой разметки.

Протокол, с помощью которого происходит обмен информацией между Web-сервером и броузером, называется HTTP (HyperText Transfer Protocol) — протокол передачи гипертекста.

Документы и разделы документов обозначаются с помощью адреса, определенного как URL (Uniform Resource Locator) — унифицированный локатор информационного ресурса. На рис. 11.1 показана схема взаимодействия основных компонентов среды Web.

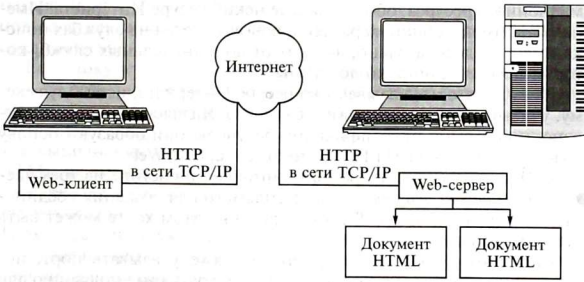


Рис. 11.1. Схема взаимодействия компонентов среды Web

Протокол HTTP действует по принципу запрос-ответ. В этом случае любая транзакция HTTP состоит из следующих этапов:

- подключение — клиент устанавливает соединение с Web-сервером;
- запрос — клиент посылает Web-серверу сообщение с запросом;
- ответ — Web-сервер посылает клиенту ответ (документ HTML);
- закрытие — соединение с Web-сервером закрывается.

Запрос HTTP включает в себя:

- заголовок, определяющий тип запроса;
- обозначение имени ресурса;
- версию протокола HTTP.

Протокол HTTP поддерживает следующие типы запросов:

GET — запрос на выборку информации (на получение информации — get);

POST — запрос на передачу информации (на отправку информации — post);

HEAD — аналогичен запросу GET, но возвращает на сервер только заголовок HTTP, а не выбранные данные;

PUT — выгружает ресурс на сервер;

DELETE — удаляет ресурс с сервера;

OPTIONS — запрашивает с сервера опции конфигурации.

URL-локатор — это строка символов, обозначающая расположение или адрес некоторого ресурса в сети Интернет, а также способ доступа к нему, т.е. URL-локатор определенным образом определяет местонахождение документа (или ресурса) в Интернете.

Идентичными понятиями URL-локатора являются идентификаторы URI и имена URN.

Идентификаторы URI (Uniform Resource Identifiers — универсальный идентификатор информационного ресурса) — это общий набор всех имен (адресов), которые относятся к ресурсам Интернет.

URN (Uniform Resource Name) — универсальное имя информационного ресурса, обозначающее некий ресурс Интернета. Имена URN имеют общий характер и основываются на службах поиска имен, а следовательно, зависят от дополнительных служб, которые не всегда широко доступны.

URL-локаторы указывают на ресурс Интернета, используя схему, учитывающую расположение ресурса. Являясь самой распространенной схемой идентификации ресурсов, они образуют основу функционирования HTTP-протокола и среды Web.

URL-локатор имеет простой синтаксис и состоит из трех частей: *обозначения протокола*, применяемого для создания соединения, *имени хоста* и *пути*, по которому на этом хосте может быть найден данный ресурс.

Кроме того, URL-локатор может также указывать порт, используя который можно подключиться к хосту (по умолчанию для HTTP-протокола используется порт 80), а также строку запроса,

которая является одним из основных способов передачи данных от клиента к серверу.

Синтаксическая структура URL-локатора имеет следующий вид:

```
<протокол>:// <узел> [:< порт > ]) / абсолютный_путь
[?параметры]
```

Здесь <протокол> описывает механизм, используемый броузером для доступа к ресурсу. Наиболее распространенными методами доступа являются: HTTP, S-HTTP (защищенный протокол HTTP), file (т.е загрузка файла с локального диска), FTP, mailto (отправка электронного письма по указанному адресу электронной почты), Gopher, NNTP и Telnet.

11.2. Статические и динамические Web-страницы

Документ HTML, хранимый в отдельном файле, является примером статической Web-страницы, т.е. его содержимое не изменяется до тех пор, пока не изменится сам файл. Содержимое динамической Web-страницы генерируется всякий раз при ее открытии.

Динамическая Web-страница может обладать следующими возможностями, которых нет у статических Web-страниц:

- реагировать на данные, вводимые пользователем с помощью броузера. Например, динамические страницы способны возвращать данные, полученные после заполнения формы или в результате выполнения запроса к базе данных;
- настраиваться по требованиям конкретного пользователя. Например, если пользователь укажет собственные предпочтения при посещении некоторого Web-узла или Web-страницы (например, область своих интересов или уровень квалификации), эту информацию можно будет сохранить, и впоследствии возвращаемая пользователю информация будет отбираться с учетом указанных предпочтений.

Если документы публикуются в динамическом режиме (например, по результатам выполнения запроса к базам данных), то сервер должен будет генерировать их в гипертекстовом формате. Следовательно, следует подготовить Web-сценарии, предназначенные для выполнения преобразования данных различного формата в HTML-формат непосредственно в процессе их формирования. Эти сценарии должны интерпретировать содержание запросов, посылаемых клиентом с помощью форм HTML, а также соблюдать формат результирующих данных, генерируемых их приложениями-владельцами (например, СУБД). Поскольку база данных является динамическим объектом, изменяющимся в результате создания, вставки, обновления или удаления пользователями сохраня-

емых в ней данных, выработка динамических Web-страниц — это гораздо более эффективный подход к работе, чем создание статических Web-страниц.

11.3. Требования к интеграции удаленных баз данных со средой Web

Развитие интернет-технологий и систем управления удаленными базами данных однозначно показало высокую эффективность создания корпоративных баз данных, доступ к которым осуществляется в среде Web.

Переход к интеграции СУБД со средой Web требует выполнения следующих условий:

- обеспечение защиты конфиденциальной информации;
- использование способов подключения удаленных пользователей, не зависящих от типов данных и программного обеспечения, в том числе с учетом развития СУБД как в настоящее время, так и в будущем;
- обеспечение возможности взаимодействия с базой данных независимо от типа используемого броузера или Web-сервера;
- обеспечение открытости архитектуры, позволяющей взаимодействовать с разнообразными системами и технологиями доступа к данным, включая поддержку транзакций, охватывающих несколько запросов HTTP, и поддержку аутентификации на уровне сеанса и приложения;
- приемлемая производительность;
- минимальные требования к администрированию;
- наличие набора высокоуровневых инструментов разработки, позволяющих относительно просто и быстро создавать, внедрять в эксплуатацию и сопровождать новые приложения.

Выполнение данных условий связано также с *оптимизацией архитектуры баз данных*.

Как уже говорилось, для осуществления интеграции баз данных со средой Web эффективна трехуровневая архитектура.

В этой архитектуре роль клиента выполняет броузер, роль сервера приложений — Web-сервер, а сервером базы данных являются традиционные СУБД (SQLServer, Oracle и др.).

В заключение сформулируем преимущества интеграции СУБД в среду Web:

- достаточная простота реализации;
- независимость от типов СУБД;
- высокий уровень стандартизации процессов передачи и обработки информации;
- возможность расширения информационных систем, в том числе при создании корпоративных связей.

11.4. Методы интеграции удаленных баз данных в среду Web

Эффективность взаимодействия СУБД и среды Web достигается следующими методами:

- применением языков сценариев;
- использованием общего шлюзового интерфейса;
- формированием Cookie-файлов HTTP;
- расширением возможностей Web-сервера.

Естественно, что это далеко не полный список возможных технологий интеграции удаленных баз данных в среду Web. Рассмотрим некоторые из указанных методов.

Применение языков сценариев. Для сохранения и передачи информации в среду Web используют языки гипертекстовой разметки, например HTML. Однако этот стандартный язык, позволяющий представлять в Интернете практически все виды информации, может вызывать определенные трудности при преобразовании приложений баз данных в Web-приложения с тем же уровнем удобства работы пользователей. Это обусловлено тем, что сам по себе язык HTML не может содержать коды приложений, управляющих работой базы данных.

Для преодоления этих трудностей и применяют языки сценариев, позволяющие встраивать код сценария в HTML-код, который загружается по сети при каждом доступе к странице.

Языки сценариев позволяют создавать функции, встраиваемые в код HTML, благодаря чему и обеспечивается возможность автоматизации различных процессов доступа и манипулирования объектами баз данных.

На сегодняшний день достаточно популярными являются следующие языки сценариев: JavaScript и Jscript, VBScript, Perl, PHP. Так как подробно описать данные языки не представляется возможным, рассмотрим лишь их некоторые особенности.

JavaScript (компании Netscape) и Jscript (компании Microsoft) — это практически идентичные интерпретируемые языки сценариев.

В этих языках предполагается непосредственная интерпретация исходного кода и вставка сценария в документ HTML. При этом сценарии могут разрабатываться как для броузера, так и для сервера.

JavaScript обеспечивает доступ к объектной модели документа. Это язык программирования, который позволяет включать в HTML-страницы функции и сценарии, способные распознавать и отвечать на действия пользователей, в том числе щелчки кнопками мыши, ввод данных и перемещение с одной страницы на другую.

JavaScript по своему синтаксису аналогичен языку Java, но он не поддерживает статических типов данных и не обеспечивает строгого контроля типов данных. В отличие от присущей языку Java си-

стемы компилируемых классов, построенной на объявлениях, в языке JavaScript используется система времени выполнения на основе типов данных, включающих в себя числовые, логические и строковые значения. JavaScript дополняет язык Java новыми возможностями при создании сценариев.

VBScript — это интерпретируемый язык сценариев компании Microsoft, назначение и принципы действия которого практически идентичны языкам сценариев JavaScript и Jscript. Однако VBScript обладает синтаксисом, который больше похож на синтаксис языка Visual Basic.

Этот язык, так же как и языки JavaScript и Jscript, может выполняться и броузером, и сервером до пересылки документа броузеру.

VBScript — это процедурный язык, в котором в качестве основного элемента используются процедуры. Он произошел от языка программирования Visual Basic, получившего широкое распространение в последние годы и являющегося базовым языком сценариев пакета Microsoft Office.

Основное отличие языка VBScript от Visual Basic состоит в том, что в целях обеспечения защиты информации из него изъяты функции работы с файлами на компьютере пользователя.

PERL (Practical Extraction and Report Language) — это высокоуровневый интерпретируемый язык программирования с широким набором удобных средств обработки текста. PERL объединяет в себе средства языка C и утилит sed, awk и sh операционной системы UNIX. В настоящее время PERL является одним из наиболее широко применяемых языков для программирования серверной части приложения, который может использоваться для разработки более мощных сценариев по сравнению с обычными сценариями командного интерпретатора UNIX .

На первых порах PERL рассматривался как язык обработки данных, позволяющий перемещаться по файловой системе, просматривать и формировать отчеты с применением механизмов согласования с шаблоном и манипулирования текстом. Однако по мере дальнейшего развития в этот язык были включены механизмы создания и управления файлами и процессами подключения к сетевым базам данных.

Сначала PERL разрабатывался на платформе UNIX, но в то же время рассматривался как перспективный межплатформенный язык. К настоящему времени выпущена версия PERL (ActivePerl) для платформы Windows.

PHP (Hypertext Preprocessor — препроцессор гипертекста) представляет собой еще один широко применяемый язык сценариев с открытым исходным кодом, операторы которого могут встраиваться в код HTML. Он поддерживается многими Web-серверами, включая HTTP-сервер Apache и Internet Information Server, а также является предпочтительным языком Web-сценариев для Linux.

Назначение данного языка — обеспечение возможности разработчикам быстрого создания сценариев динамического формирования страниц. Одним из преимуществ PHP является его расширяемость, поэтому уже разработан целый ряд модулей расширения для поддержки таких функций, как подключение к базе данных, передача и прием электронной почты, а также обработка данных в коде XML.

Использование общего шлюзового интерфейса — CGI (Common Gateway Interface). CGI представляет собой набор средств передачи информации между Web-сервером и программой.

Для отображения запрашиваемого документа броузеру надо знать о нем совсем немного. После отправки требуемого URL (адреса ресурса) броузер (просмотрщик) отображает полученный ответ в том виде, в каком он был получен (без дальнейшей обработки). Для того чтобы броузер смог отличить одни полученные им компоненты от других, сервер предоставляет специальные коды, присваиваемые отдельным элементам сообщения на основе спецификации MIME (Multipurpose Internet Mail Extensions). Эта спецификация, в частности, позволяет броузеру установить, что данный файл является либо графическим и его следует отобразить на экране, либо архивным и тогда его следует (при необходимости) сохранить на диске.

ЗадачаWeb-сервера заключается лишь в том, чтобы отправить документы броузеру и сообщить, к какому типу они относятся. Кроме того, сервер должен при необходимости выполнить запуск других программ.

Если сервер распознает, что полученный URL указывает на некий файл, он отправляет броузеру содержимое этого файла. Если же сервер распознает, что поступивший URL указывает на некоторую программу (или сценарий), то посылает броузеру результат выполнения этого сценария, представленный в виде содержимого некоторого файла.

CGI определяет способ взаимодействия сценариев с Web-серверами. Сценарием CGI называется любой сценарий, который может принимать и передавать данные в соответствии со спецификацией CGI.

Таким образом, совместимый со спецификацией CGI сценарий может повсеместно использоваться для предоставления информации независимо от типа конкретного сервера.

11.5. Генерация Web-страниц визуальными средствами Microsoft Access

Ранее рассматривались основные компоненты организации доступа к данным средствами Access. Теперь дадим краткое описа-

ние компонентов этой СУБД, позволяющих интегрировать ее базы данных в среду Web.

В Access, начиная с версии 2000, предусмотрены три программы-мастера, предназначенные для автоматической генерации HTML-страниц на основе таблиц, запросов, форм или отчетов, представленных в базе данных:

- мастер создания статических страниц;
- мастер создания динамических страниц на основе технологии ASP;
- мастер создания динамических страниц, формируемых с помощью страниц доступа к данным.

Создание статических страниц. Этот метод позволяет пользователю экспортировать данные в формате HTML. Применяемые при этом функции являются несложными, но обладают очевидным недостатком, связанным с тем, что содержимое HTML-страницы может быстро устаревать и ее требуется повторно формировать при каждом изменении информации в таблице (таблицах) базы данных. При создании подобных страниц используется стандартный язык HTML, обрабатывать который способен любой броузер. Пользователь имеет возможность до определенной степени управлять внешним видом формируемой Web-страницы с помощью так называемых шаблонов HTML — файлов, которые состоят из операторов HTML, описывающих компоновку страницы. Шаблоны позволяют вставить в код страницы логотип компании, изображения и другие элементы.

Создание динамических страниц на основе технологии ASP. Этот метод позволят пользователю экспортировать данные в виде файла с расширением .asp наWeb-сервер, указав имя текущей базы данных, идентификатор и пароль пользователя для подключения к базе данных, а также URL Web-сервера, на котором должен быть записан этот файл ASP.

Создание динамических страниц, формируемых с помощью страниц доступа к данным. Страницы доступа к данным представляют собой Web-страницы, связанные непосредственно с информацией, находящейся в базе данных. Такие страницы, кроме того, что они хранятся во внешних файлах, а не в базе данных или в составе файлов проекта приложения для базы данных, могут использоваться как формы Access. Хотя эти страницы и могут использоваться в приложении Access, в основном они предназначены для просмотра на броузере. Страницы доступа к данным разрабатываются с применением языка DHTML (Dynamic HTML) — расширения языка HTML, которое позволяет включать динамические объекты в состав Web-страницы.

В отличие от файлов ASP страница доступа к данным создается в среде Access с помощью программы-мастера или представления Design (Конструктор). В этом случае применяются в основном та-

кие же инструментальные средства, как при создании форм. Однако при использовании станицы доступа к данным в качестве броузера должен применяться Интернет Explorer 5.0 или более поздних версий, поскольку только эти программы позволяют работать с такими страницами.

Контрольные вопросы

1. Где и когда закладывались основы сети Интернет?
2. Что означает термин TCP/IP?
3. Каково назначение следующих протоколов передачи информации: FTP, SMTP, Telnet, DNS, POP?
4. В чем состоит различие между внутренними и внешними сетями?
5. Что представляет собой среда Web?
6. Назовите программные пакеты, выполняющие функции Web-серверов.
7. Назовите программные пакеты, выполняющие функции Web-клиентов.
8. Каково назначение протокола HTTP?
9. Какие типы запросов поддерживает протокол HTTP?
10. Каковы основные требования к интеграции удаленных баз данных со средой Web?
11. Охарактеризуйте следующие методы взаимодействия СУБД со средой Web: применение языков сценариев, использование общего шлюзового интерфейса, формирование Cookie-файлов HTTP.
12. Какие три программы-мастера имеются в последних версиях СУБД Access, предназначенные для автоматической генерации HTML-страниц?

ЧАСТЬ V

АДМИНИСТРИРОВАНИЕ И ЭКСПЛУАТАЦИЯ УДАЛЕННЫХ БАЗ ДАННЫХ

ГЛАВА 12

ЗАЩИТА ИНФОРМАЦИИ И УПРАВЛЕНИЕ ДОСТУПОМ К ДАННЫМ

12.1. Основные проблемы и способы защиты баз данных

Рассмотрим общие проблемы организации доступа к информации и ее защиты в удаленных базах данных.

Защита баз данных предполагается против любых предумышленных или непредумышленных угроз и заключает в себе различные организационные меры, программные и технические средства.

Понятие защиты применимо не только к информации, хранящейся в базах данных, необходимость защиты информации может возникать и в других частях информационных систем, что, в свою очередь, обусловит защиту и самой базы данных. Следовательно, защита базы данных является комплексной задачей и должна охватывать все коммуникационные системы ЛВС предприятия, включая оборудование, программное обеспечение, персонал и собственно данные.

База данных представляет собой важнейший корпоративный ресурс, который должен быть надлежащим образом защищен с помощью соответствующих средств контроля.

Рассмотрим способы защиты базы данных от следующих потенциальных опасностей:

- похищение и фальсификация данных;
- утрата конфиденциальности (нарушение тайны);
- нарушение неприкосновенности личных данных;
- утрата целостности;
- потеря доступности.

Это основные направления, по которым руководство предприятия должно принимать меры, обеспечивающие снижение степени риска потерь или повреждения данных.

Исходя из сказанного любая угроза, нарушающая функционирование информационной системы, должна рассматриваться как ситуация, направленная на катастрофические результаты работы предприятия.

Таблица 12.1

Возможные опасности для информационных систем

Опасность	Похищение и фальсификация данных	Утрата конфиденциальности	Нарушение неприкосновенности личных данных	Утрата целостности	Потеря доступности
Использование прав доступа другим лицом	+	+	+	—	—
Несанкционированное изменение или копирование данных	+	—	—	+	—
Изменение программ	+	—	—	+	+
Непродуманные организационные инструкции, допускающие смешивать в одном документе секретную и несекретную информацию	+	+	+	—	—
Несанкционированное подключение к кабельным сетям	+	+	+	—	—
Несанкционированный ввод данных	+	+	+	+	—
Отказ аппаратных и программных средств	—	+	+	+	+
Электронные помехи	—	—	—	+	+
Физическое повреждение оборудования и коммуникаций	—	—	—	+	+

В табл. 12.1 показаны примеры возможных опасностей для информационных систем.

Проблемы обеспечения безопасности баз данных можно подразделить на две категории: технологическую и организационную. Однако в реальной практике эти категории неразрывны.

12.2. Технологические методы защиты информации

Рассмотрим основные факторы, определяющие технологическую безопасность информационных систем.

Технологическая безопасность информационных систем определяется как *алгоритмическая и программно-аппаратная*, однако для

краткости будем использовать термин *технологическая безопасность*, или *безопасность*.

Проблемы обеспечения технологической безопасности информационных систем можно свести к следующим аспектам:

- обеспечение непрерывности и корректности функционирования систем, от которых зависит безопасность людей и экологической обстановки;
- обеспечение защиты имущественных прав граждан, предприятий и государства в соответствии с требованиями гражданского, административного и хозяйственного кодексов (включая защиту секретов и интеллектуальной собственности);
- обеспечение защиты гражданских прав и свобод, гарантированных действующим законодательством (включая право на доступ к информации).

Следует еще раз отметить, что важнейшим назначением любой информации является то, что она *служит основой для принятия оптимальных решений практически в любых сферах человеческой деятельности.*

Требования по безопасности информационных систем различных предприятий могут существенно отличаться, однако они всегда должны обеспечивать следующие три основные свойства информации:

целостность, т.е. информация, на основе которой принимаются решения, должна быть достоверной и точной, в том числе защищенной от возможных непреднамеренных и злоумышленных искажений;

доступность, т.е. информация и соответствующие службы администрирования данных должны быть доступны и готовы к работе всегда, когда в них возникает необходимость;

конфиденциальность, т.е. конфиденциальная (засекреченная) информация должна быть доступна только тому, кому она предназначена.

Обеспечение защиты информации включает в себя:

- разработку показателей, характеризующих технологическую безопасность информационных систем;
- разработку требований к архитектуре баз данных;
- наличие трудовых и материальных ресурсов;
- разработку организационных мероприятий для исключения влияния внутренних и внешних дестабилизирующих факторов;
- разработку методов и средств, предотвращающих влияние дефектов программ и данных, в том числе разработку компьютерных экспертных систем оценки качества программных продуктов.

Показатели технологической безопасности информационных систем. Наиболее полно безопасность информационной системы характеризует *ущерб*, возможный при проявлении конкретной угрозы безопасности.

Однако описание и расчет возможного ущерба в достаточно общем виде является сложной задачей. Данная проблема в некотором смысле идентична проблеме оценки эффективности и надежности сложных технических систем, основанных на вероятностных методах.

Понятия *характеристика степени безопасности* и *показатели надежности* информационных систем достаточно близки. Различие состоит лишь в том, что показатели надежности учитывают *все возникающие отказы* при эксплуатации баз данных, а в характеристиках безопасности должны учитываться только *отказы, повлиявшие на безопасность системы.*

В соответствии с теорией надежности работоспособным называют состояние информационной системы (программных, аппаратных и трудовых ресурсов), при котором она способна выполнять заданные функции.

Показатели надежности баз данных оцениваются по следующим критериям: устойчивость, восстанавливаемость, коэффициент готовности.

Устойчивость (живучесть) — критерий, наиболее широко характеризующий способность информационной системы к безотказной работе при наличии сбоев и отказов программных и аппаратных средств, что обеспечивается:

- эффективным контролем за доступом к данным;
- обеспечением высокой степени конфиденциальности и целостности данных;
- контролем данных, поступающих из внешней среды.

Восстанавливаемость — критерий, определяемый временем и полнотой восстановления функционирования программ после перезапуска в случаях сбоя или отказа.

Коэффициент готовности — критерий, характеризующий степень вероятности восстановления системы в любой произвольный момент времени. Значение коэффициента готовности соответствует доле времени полезной работы системы на достаточно большом интервале, содержащем отказы и восстановления.

Приведенные критерии используются в основном при испытании информационных систем и на завершающих фазах комплексной отладки.

Требование к архитектуре информационных систем. Основное требование сводится к следующему: *архитектура должна быть достаточно гибкой и допускать наращивание функций и ресурсов информационной системы без коренных структурных изменений, например за счет развития используемых программных и аппаратных средств.*

Для выполнения этого требования необходимо наличие *программной* и *информационной избыточности* системы в виде ресурсов внешней и внутренней памяти ЭВМ.

Кроме того, для функционирования средств защиты необходима *временная избыточность вычислительных ресурсов*, обеспечиваемая высокой производительностью аппаратных средств ЛВС предприятия.

Все виды избыточности вычислительных ресурсов при обеспечении технологической безопасности используются для генерации тестовых наборов или хранения тестов контроля работоспособности и целостности ИС и БД при функционировании ИС, а также для оперативного контроля обнаружения и анализа дефектов исполнения программ.

Средства генерации тестов предназначены для подготовки исходных данных при проверке различных режимов функционирования информационной системы. Минимальный состав средств имитации может передаваться пользователям для контроля рабочих версий ИС в реальном времени и входить в комплект поставки каждой пользовательской версии. Для более глубоких испытаний версий и локализации ошибок целесообразно создавать комплексы средств имитации внешней среды высшего уровня, используемые специалистами по испытаниям и сертификации. Часть этих средств может применяться также в качестве средств имитации среды нижнего уровня (пользовательских) для обеспечения полного повторения ситуаций, при которых обнаружены аномалии функционирования ИС.

Средства генерации, упорядочения и каталогизации тестовых наборов должны обеспечивать возможность многократного использования тестов в течение жизненного цикла информационной системы. Для эффективного использования тестов необходима система управления базой данных, обеспечивающая их накопление и хранение с тщательно продуманной идентификацией и каталогизацией. Система каталогизации должна обеспечивать достаточно простой и надежный поиск имеющихся тестов, а также достоверное выявление тестов, отсутствующих среди сохраняемых.

Средства оперативного (встроенного) контроля процесса исполнения программ должны непрерывно контролировать промежуточные и результирующие данные или включаться только по запросу при обнаружении сомнительных результатов. Они также должны обеспечивать получение информации о состоянии переменных в процессе решения конкретных задач и маршрутах исполнения программ, в которых нарушаются некоторые заданные условия. Создаваемые для эксплуатации методики и инструкции позволяют пользователям достаточно квалифицированно осуществлять диагностику состояния информационной системы. В настоящее время предприятия все чаще прибегают к созданию компьютерных экспертных систем.

Методы обеспечения технологической безопасности информационных систем. В табл. 12.1 были приведены возможные опасно-

сти для информационных систем. Рассмотрим основные уязвимые объекты для неумышленных угроз.

Такими объектами являются:

- динамический вычислительный процесс обработки данных, автоматизированной подготовки решений и выработки управляющих воздействий;
- информация, накопленная в базах данных;
- объектный код программ, исполняемых вычислительными средствами в процессе функционирования ИС;
- информация, выдаваемая потребителям и на исполнительные механизмы.

Возможные непредумышленные дестабилизирующие факторы можно подразделить на внешние и внутренние.

Внутренние источники угроз безопасности ИС:

- системные ошибки при разработке технического задания на разработку удаленных баз данных;
- алгоритмические ошибки проектирования и эксплуатации баз данных;
- ошибки программирования;
- недостаточная эффективность используемых методов и средств оперативной защиты программ и данных;

Внешние источники угроз безопасности ИС:

- ошибки оперативного и обслуживающего персонала в процессе эксплуатации баз данных;
- искажения в каналах телекоммуникации информации, поступающей от внешних источников и передаваемой потребителям, а также недопустимые изменения характеристик потоков информации;
- сбои и отказы аппаратуры;
- выход изменений состава и конфигурации ИС за пределы, проверенные при испытаниях или сертификации.

Полное устранение перечисленных угроз безопасности ИС принципиально невозможно. Следовательно, необходимо выявлять факторы, определяющие эти угрозы, и создавать методы и средства, уменьшающие их влияние на безопасность баз данных.

Современные технологии разработки удаленных баз данных определяют следующие методы и средства, позволяющие с максимальным эффектом обеспечить технологическую безопасность ИС:

- разработка баз данных в полном соответствии с методологией их проектирования (см. гл. 2, 7);
- систематическое тестирование программ управления базами данных на всех этапах жизненного цикла;
- применение экспертных систем в процессе сертификации СУБД и сдачи их в эксплуатацию;
- применение программно-аппаратных методов защиты информации в критических ситуациях;

• физическое уничтожение информации в критических ситуациях.

Комплексное скоординированное применение указанных методов и средств позволяет исключить возможные угрозы безопасности ИС или значительно ослабить их влияние.

Далее рассмотрим несколько подробнее программно-аппаратные методы защиты информации и методы ее физического уничтожения в критических ситуациях.

К программно-аппаратным методам защиты информации в базах данных относятся авторизация пользователей, применение представлений, резервное копирование и восстановление, шифрование и создание массивов независимых дисковых накопителей.

Авторизация пользователей

Авторизация пользователей — это представление прав (привилегий), позволяющих их владельцу иметь законный доступ к информации в базах данных или к системе управления базами данных, или к отдельным ее объектам.

В данном определении термин *владелец* означает физическое лицо или программу, а термин *объект* — любой компонент СУБД, который может быть создан в рамках конкретной системы (таблица базы данных, представление, приложение, триггер и т.п.).

Программы, обеспечивающие авторизацию пользователей, называют *средствами управления доступом*.

Процесс авторизации подразумевает необходимость аутентификации пользователя и предоставления ему привилегий и прав владения.

Аутентификация. Способ определения того, что пользователь является тем, за кого себя выдает, называется аутентификацией.

За предоставление доступа к компьютерной системе обычно отвечает системный администратор, в обязанности которого входит создание учетных записей пользователей.

Каждому пользователю присваивается уникальный идентификатор, используемый операционной системой для определения «кто есть кто». С каждым идентификатором связан определенный пароль, выбираемый пользователем и известный операционной системе.

При регистрации пользователь должен предоставлять системе свой пароль для выполнения аутентификации, т.е. определения, является ли он тем, за кого себя выдает.

Подобная процедура позволяет организовать контролируемый доступ к компьютерной системе, но не обязательно предоставляет право доступа к СУБД или какой-либо прикладной программе.

Для получения пользователем права доступа к СУБД может применяться отдельная процедура.

Ответственность за предоставление прав доступа к СУБД обычно несет администратор базы данных, в обязанности которого входит создание отдельных идентификаторов пользователей для работы с конкретной базой данных.

В одних СУБД ведется список идентификаторов пользователей и связанных с ними паролей, отличающийся от аналогичного списка, поддерживаемого операционной системой, а в других — ведется список, записи которого сверяются с записями списка пользователей операционной системы с учетом текущего регистрационного идентификатора пользователя. Это предотвращает возможность регистрации пользователя в среде СУБД под идентификатором, отличным от того, который он использовал при регистрации в системе.

Привилегии. Как только пользователь получает право доступа к СУБД, ему автоматически предоставляются различные привилегии, связанные с его идентификатором.

В частности, привилегии могут включать в себя разрешение на доступ к определенным базам данных, таблицам, представлениям и индексам, а также разрешение на создание этих объектов или же право вызывать на выполнение различные утилиты СУБД.

Привилегии предоставляются пользователям лишь для того, чтобы они могли выполнять задачи, которые входят в круг их непосредственных должностных обязанностей. Предоставление излишних привилегий может привести к нарушению защищенности баз данных.

Некоторые типы СУБД функционируют как *закрытые системы*, и их пользователям помимо разрешения на доступ к самой СУБД требуется иметь отдельные разрешения на доступ к конкретным ее объектам. Эти разрешения выдаются администратором базы данных с разрешения владельцев соответствующих объектов системы.

В отличие от закрытых *открытые системы* по умолчанию предоставляют пользователям, прошедшим аутентификацию, полный доступ ко всем объектам базы данных.

Стандарт ISO/EC9075:2003 определяет следующий набор привилегий языка SQL:

SELECT — право выбирать данные из таблицы;

INSERT — право вставлять в таблицу новые строки;

UPDATE — право изменять данные в таблице;

DELETE — право удалять строки из таблицы;

REFERENCES — право ссылаться на столбцы указанной таблицы в описании требований поддержки целостности данных.

Привилегии INSERT и UPDATE могут ограничиваться отдельными столбцами таблицы и в этом случае пользователь может модифицировать значения только указанных столбцов.

Привилегия REFERENCES также может распространяться только на отдельные столбцы таблицы, что позволит использовать их имена в формулировках требований защиты целостности данных (например, в конструкциях CHECK FOREIGN REY), входящих в определения других таблиц, тогда как применение для подобных целей остальных столбцов будет запрещено.

Когда пользователь с помощью оператора CREATE TABLE создает новую таблицу, он автоматически становится ее владельцем и получает по отношению к ней полный набор привилегий.

Остальные же пользователи сначала не имеют никаких привилегий в отношении вновь созданной таблицы и для обеспечения им доступа к этой таблице используется оператор GRANT.

Если пользователь создает представление с помощью оператора CREATE VIEW, он автоматически становится владельцем этого представления, однако совсем необязательно, что получает по отношению к нему полный набор прав.

При создании представления пользователю достаточно иметь привилегию SELECT для всех входящих в данное представление таблиц и привилегию REFERENCES для всех столбцов, упоминаемых в определении этого представления.

Привилегии INSERT, UPDATE, DELETE в отношении созданного представления пользователь получит только в том случае, если он имеет соответствующие привилегии в отношении всех используемых в представлении таблиц.

Предоставление привилегий другим пользователям. Оператор GRANT используется для предоставления привилегий определенным пользователям в отношении поименованных объектов базы данных с разрешения ее владельца.

Оператор GRANT имеет следующий формат:

```
GRANT {PrivilegeList | ALL PRIVILEGES}
ON ObjectName
TO {AutohrizationIdList | PUBLIC}
[WITH GRANT OPTION]
```

Параметр PrivilegeList представляет собой список, состоящий из одной или более привилегий, разделенных запятыми:

```
INSERT [(ColumnName [, …])]
UPDATE [(ColumnName [, …])]
REFERENCES [(ColumnName [, …])]
USAGE [(ColumnName [, …])]
```

Кроме того, для упрощения в операторе GRANT можно указать ключевое слово ALL PRIVILEGES, что позволит предоставить указанному пользователю все шесть существующих привилегий без необходимости их перечисления.

В этом операторе можно также указать ключевое слово PUBLIC, означающее предоставление доступа указанного типа не только всем существующим пользователям, но и всем тем пользователям, которые будут определены в базе данных впоследствии.

Параметр ObjectName может представлять собой имя таблицы базы данных, представления, домена, набора символов, проверки или транзакции.

Конструкция WITH GRANT OPTION позволяет всем пользователям, указанным в списке параметра AutohrizationIdList, передавать другим пользователям все предоставленные им в отношении указанного объекта привилегии. Если эти пользователи, в свою очередь, передадут собственные полномочия другим пользователям с указанием конструкции WITH GRANT OPTION, то последние также получат право передавать свои полномочия. Если же эта конструкция не будет указана, получатель привилегии не сможет передавать свои права другим пользователям. Таким образом, владелец объекта может четко контролировать, кто получил право доступа к принадлежащему ему объекту и какие полномочия предоставлены этому лицу.

Приведем пример предоставления пользователю с идентификатором *Administrator* всех привилегий доступа к таблице MK (Маршрутная карта):

```
GRANT ALL PRIVILEGES
ON MK
TO Administrator WITH GRANT OPTION;
```

В результате выполнения этого примера пользователь с идентификатором *Administrator* получает право выбирать данные из таблицы MK, а также вставлять, обновлять или удалять из нее строки. Кроме того, пользователь *Administrator* может ссылаться на таблицу MK и все ее столбцы в любой таблице, создаваемой им впоследствии. Так как в данном примере присутствует конструкция WITH GRANT OPTION, пользователь *Administrator* сможет передавать полученные им привилегии по своему усмотрению другим пользователям.

Приведем пример предоставления пользователям с идентификаторами *Texnolog* и *Konstruktor* только привилегий SELECT и UPDATE на столбец *NaimOper* таблицы *MK* (*Маршрутная карта*):

```
GRANT SELECT, UPDATE (Naim Oper)
ON MK
TO Texnolog, Konstruktor ;
```

Поскольку в последнем примере отсутствует конструкция WITH GRANT OPTION, указанные пользователи не смогут передать полученные привилегии другим пользователям.

Для отмены предоставленных пользователям привилегий используют оператор REVOKE, который имеет следующий формат:

```
REVOKE [GRANT OPTION FOR] {PrivilegeList | ALL
PRIVILEGES}
ON ObjectName
FROM {AutohrizationIdList | PUBLIC} [RESTRICT |
CASCADE]
```

Здесь ключевые слова ALL PRIVILEGES означают, что для указанного пользователя отменяются все привилегии, предоставленные ему ранее тем пользователем, который ввел данный оператор. Необязательная конструкция GRANT OPTION FOR позволяет для всех привилегий, переданных в исходном операторе GRANT конструкции WITH GRANT OPTION, отменять возможность их передачи. Назначение ключевых слов RESTRICT и CASCADE аналогично назначению, которое они имеют в операторе DROP TABLE (см. гл. 8).

Применение представлений

Технология создания пользовательских представлений рассматривалась в гл. 8, здесь же приведем аспекты данного объекта баз данных с позиции защиты информации.

Напомним, что представление является как бы виртуальным отношением (динамической таблицей) базы данных, которое создается в результате запроса пользователя и доступно только самому пользователю.

Механизм представлений служит достаточно эффективным средством защиты баз данных от несанкционированного доступа, поскольку он доступен только автору представления.

Резервное копирование и восстановление

Резервное копирование и восстановление — это периодически выполняемая процедура получения копий базы данных, файлов журналов, программ на носителях, хранящихся отдельно от системы.

Любая современная СУБД должна предоставлять администратору баз данных средства резервного копирования, позволяющие восстанавливать ее в случае разрушения.

Рекомендуется создание резервных копий базы данных и файлов ее журналов с некоторой установленной периодичностью, а также организация хранения этих копий в местах, обеспеченных необходимой защитой, что будет рассмотрено далее.

СУБД должна предоставлять средства ведения системного журнала, в котором будут фиксироваться сведения обо всех изменениях состояния базы данных и ходе выполнения текущих транзакций, что необходимо для эффективного ее восстановления в случае отказа системы.

Преимущества использования такого журнала заключаются в том, что в случае нарушения работы системы или отказа базу данных можно будет восстановить до последнего известного согласованного состояния, воспользовавшись ее последней созданной резервной копией и оперативной информацией, содержащейся в файле журнала.

Если же в отказавшей системе функция ведения системного журнала не использовалась, базу данных можно будет восстановить только до состояния, зафиксированного в последней резервной копии. Все изменения, внесенные в базу данных после создания последней резервной копии, будут потеряны.

Определенный вклад в защищенность системы вносит поддержка целостности данных, поскольку она предотвращает угрозу получения ошибочной информации.

Шифрование

Шифрование — это процесс преобразования данных с применением специального алгоритма, в результате чего данные становятся недоступными для считывания любой программой, не имеющей соответствующего ключа дешифрования.

Шифрование необходимо, если в системе с базой данных содержится важная конфиденциальная информация.

Шифрование имеет смысл применять также для защиты информации при передаче ее по линиям связи.

Существует множество различных технологий шифрования данных, которые подразделяют на две категории: *необратимые* и *обратимые*.

Необратимые технологии, как и следует из их названия, не позволяют восстанавливать исходные данные, поэтому чаще применяют обратимые технологии.

Для организации защищенной передачи данных по незащищенным сетям используют системы шифрования, включающие в себя следующие компоненты:

- алгоритм шифрования;
- алгоритм дешифрования;
- ключ шифрования, т.е. ключ, предназначенный для шифрования текстовой информации;
- ключ дешифрования, т.е. ключ, предназначенный для дешифрования зашифрованного текста.

Некоторые системы шифрования, называемые *симметричными*, используют один и тот же ключ как для шифрования, так и для дешифрования.

Одной из наиболее распространенных является система DES (Data Encryption Standart), в которой используется стандартный алгоритм шифрования, разработанный фирмой IBM. В этой системе для шифрования и дешифрования применяется один и тот же ключ, который должен храниться в секрете, хотя сам алгоритм, предусматривающий преобразование каждого 64-битового блока обычного текста с использованием 56-битового ключа шифрования, не является секретным.

В другой системе PGP POP (Pretty Good Privacy) используется 128-битовый симметричный алгоритм, применяемый для шифрования блоков передаваемых данных переменной длины.

Системы шифрования, предусматривающие применение различных ключей для шифрования и дешифрования информации, называют *асимметричными*.

Симметричные системы шифрования более быстродействующие, чем асимметричные, однако на практике их применяют в сочетании друг с другом.

Создание массивов независимых дисковых накопителей

Очевидно, что к аппаратному обеспечению, на котором эксплуатируется СУБД, предъявляются высокие требования по отказоустойчивости, т.е. СУБД должна продолжать работать даже при отказе отдельных аппаратных компонентов. Для этого необходимо иметь избыточные компоненты, которые могут быть объединены в систему, сохраняющую работоспособность при отказе одного или нескольких устройств. К числу аппаратных средств, к которым предъявляются высокие требования по отказоустойчивости, относятся дисковые накопители, дисковые контроллеры, процессоры, источники питания и вентиляторы охлаждения. Дисковые накопители являются наиболее уязвимыми среди всех аппаратных компонентов и характеризуются самыми низкими показателями непрерывной работы между отказами.

Одним из методов повышения отказоустойчивости дисковых накопителей является применение RAID-технологии. Первоначально аббревиатура RAID расшифровывалась как Redundant Array of Inexpensive Disks (массив недорогих дисковых накопителей с избыточностью), но в дальнейшем слово «Inexpensive» в этой аббревиатуре заменили на Independent (независимый).

RAID-массив представляет собой массив дисковых накопителей большого объема, состоящий из нескольких независимых дис-

ков. Совместное функционирование таких дисков обеспечивает повышение их надежности и производительности.

Повышение производительности осуществляется вследствие *полосового распределения данных* на дисках, т.е. распределения данных по сегментам, представляющим собой области дискового пространства одного размера, называемого *единицей полосового распределения*. Сегменты распределяются по нескольким дискам и обеспечивают прозрачный доступ к данным. В результате такой массив данных становится аналогичным одному крупному быстродействующему диску. Полосовое распределение соответствует размещению информации как бы по нескольким дискам меньшего объема, в результате чего обеспечивается повышение производительности ввода-вывода, поскольку операция выполняется на разных дисках одновременно.

Полосовое распределение данных позволяет равномерно распределять нагрузку между дисками.

Повышенная надежность RAID-массива обеспечивается также благодаря дублированию данных в виде *зеркальных копий*, которые и создают как бы избыточную информацию.

В RAID-массивах используются различные методы повышения производительности и надежности, получившие название уровней RAID.

- RAID 0 — неизбыточный массив. На этом уровне не применяется дублирование данных и поэтому обеспечивается наивысшая производительность, поскольку не приходится копировать по дискам обновляемые записи. Полосовое распределение данных осуществляется на уровне дисковых блоков.
- RAID 1 — массив с зеркальным отображением. На этом уровне создаются две идентичные (зеркальные) копии данных на разных дисках. Этот вариант организации хранения данных является наиболее дорогостоящим.
- RAID 0+1 — неизбыточный массив с зеркальным отображением. В данном варианте применяется сочетание методов полосового распределения и зеркального отображения данных.
- RAID 2 — массив с применением кодов корректировки ошибок, хранящихся во внешней памяти. На этом уровне единицей полосового распределения является 1 бит.
- RAID 3 — массив, обеспечивающий контроль четности с чередованием битов. На этом уровне предусматривается хранение избыточных данных, представляющих собой информацию контроля четности, на отдельном диске массива. Данная информация может применяться для восстановления данных, хранящихся на других дисках, в случае их отказа. На этом уровне используется меньший объем пространства внешней памяти по сравнению с уровнем RAID 1.
- RAID 4 — массив, обеспечивающий контроль четности с чередованием блоков. На этом уровне единицей полосового распре-

деления является блок диска, а блоки с информацией контроля четности хранятся на отдельном диске. В этом случае при отказе отдельных дисков с данными блок контроля четности может применяться в сочетании с соответствующими блоками других дисков для восстановления потерянных данных.

- RAID 5 — массив, обеспечивающий контроль четности с чередованием блоков и распределением информации контроля четности. На этом уровне информация контроля четности применяется в качестве избыточной и обеспечивающей восстановление первоначальных данных по такому же принципу, как в массиве RAID 3. При этом данные контроля четности распределяются с помощью метода полосового распределения по всем дискам так же, как происходит распределение исходных данных, что позволяет устранить узкое место, возникающее при хранении всей информации контроля четности на одном диске.
- RAID 6 — массив с избыточностью P+Q. Этот уровень аналогичен уровню RAID 5, но предусматривает хранение дополнительных избыточных данных для защиты от отказа сразу нескольких дисков. При этом вместо информации контроля четности используются коды исправления ошибок.

Корпорация Oracle рекомендует использовать для файлов журнала восстановления уровень RAID 1, а для файлов базы данных — уровень RAID 5.

12.3. Дисковое хранилище с системой уничтожения данных

Одним из технологических методов защиты конфиденциальной информации является физическое уничтожение данных в критических ситуациях, которые возможны в корпоративных базах данных с секретной информацией (например, банковских информационных системах, конструкторских и технологических базах данных предприятий оборонного комплекса и др.).

Для мгновенного уничтожения информации с магнитных носителей используют отдельно стоящую или встроенную в корпус компьютера систему.

В настоящее время оптимальным подходом, обеспечивающим уничтожение информации без уничтожения носителя, является использование физических методов, заключающихся в перестройке структуры магнитного материала рабочих поверхностей носителя.

Для уничтожения информации на магнитной пластине накопителя на жестком магнитном диске (НЖМД) необходимо устранить неоднородность вектора намагниченности участков его рабочих поверхностей, несущих информацию о предшествующих записях.

Изменение структуры поля вектора намагниченности магнитного материала может быть выполнено несколькими принципиально различными способами.

1. Быстрое нагревание материала рабочего слоя носителя до точки потери намагниченности носителя (точки Кюри).

2. Размагничивание рабочих поверхностей носителя.

3. Намагничивание рабочих поверхностей носителя до максимально возможных значений (до насыщения).

4. Комбинированный, т.е. нагревание и намагничивание либо нагревание и размагничивание.

Первый способ основывается на одном из важных эффектов магнетизма: при нагревании ферромагнетика до температуры, превышающей точку Кюри, интенсивность теплового движения атомов становится достаточной для разрушения его самопроизвольной намагниченности, и он становится парамагнетиком. Следовательно, при такой температуре ферромагнитный материал рабочего слоя теряет свою остаточную намагниченность, и все следы ранее записанной информации гарантированно уничтожаются.

Температура, соответствующая точке Кюри, у большинства ферромагнитных материалов рабочего слоя носителей информации составляет сотни градусов. При этом надо учитывать, что каждый производитель НЖМД держит в секрете материал основы и состав ферромагнитного покрытия. Вероятнее всего, наиболее уязвимыми для температурных воздействий компонентами рабочего слоя и основы НЖМД окажутся связующие материалы органической природы. В этом случае при нагревании до высоких температур НЖМД выйдет из строя по причине плавления элементов конструкции, имеющих температуру плавления или деформации меньше точки Кюри.

Второй способ заключается в размагничивании ферромагнетика в медленно убывающем переменном магнитном поле.

В случае с НЖМД возникают трудности, связанные с большой коэрцитивной силой (остаточной намагниченностью) ферромагнитного покрытия диска. Получение сильных стационарных полей в зазорах электромагнитов ограничено индукцией насыщения магнитопровода, составляющей около 2 Тл.

Использование мощного постоянного магнита для композиции самарий — кобальт или сходных по характеристикам композиций на основе лантаноидов вызывает технологические трудности. Расчеты показывают, что для создания равномерного поля в воздушном зазоре при размещении в нем НЖМД с максимальным размером до 87,5 мм (с учетом накопителей, используемых на серверах) необходим постоянный магнит сложной формы с концентратором поля. Технологические возможности современной промышленной базы позволяют создать такой магнит, одна-

ко выпуск единичного экземпляра или малой серии этих магнитов экономически нецелесообразен.

Третий способ основан на представлении внешнего магнитного поля НЖМД как аналога поля, создаваемого магнитными головками при записи. Если напряженность внешнего поля будет превышать напряженность поля, создаваемого магнитными головками, на значение, при котором происходит магнитное насыщение материала поверхности диска, то все магнитные домены будут переориентированы по направлению этого внешнего поля и вся информация на НЖМД будет уничтожена.

Для ферромагнетиков характерен гистерезис при перемагничивании внешним магнитным полем. Под воздействием внешнего магнитного поля происходит ориентация элементарных магнитных полей, создаваемых круговым движением электронов в атомах ферромагнетика. В результате увеличиваются размеры магнитных доменов, ориентированных по направлению внешнего поля. После прекращения внешнего воздействия изменения размеров и ориентации магнитных доменов частично сохраняются и, следовательно, появляется остаточная намагниченность вещества. Именно эту остаточную намагниченность материала носителя регистрируют затем устройства, считывающие записанную информацию.

Физические основы процессов, происходящих в накопителе под влиянием внешнего магнитного поля, связаны с его конструктивными особенностями и спецификой применяемых материалов. Ввиду того что характеристики материала, из которого изготавливаются покрытия поверхностей современных НЖМД, как правило, фирмами-производителями не разглашаются, оценку требуемой напряженности намагничивающего поля приходится рассчитывать с некоторым запасом: значение напряженности поля стирания для магнитной ленты при условии однопроходного воздействия должно превышать значение коэрцитивной силы в 4 раза.

Импульсные намагничивающие установки удовлетворяют указанному требованию, обеспечивая:

- возможность создания сильных намагничивающих полей с малыми энергетическими затратами;
- кратковременность воздействия импульсного поля на образец;
- возможность помещения НЖМД целиком в камеру намагничивания;
- возможность применения простых индукторных систем разомкнутого типа без магнитопровода;
- формирование магнитного поля необходимой направленности.

Наиболее простыми являются импульсные источники тока для намагничивающих устройств, в которых энергия сети и емкост-

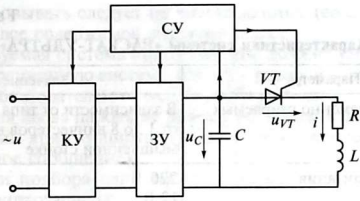


Рис. 12.1. Структурная схема устройства намагничивания импульсного типа

ного накопителя поступает в виде импульса непосредственно в индуктор.

Структурная схема устройства намагничивания импульсного типа приведена на рис. 12.1.

В этом устройстве емкостной накопитель, представляющий собой батарею конденсаторов с емкостью C, заряжается до необходимого напряжения от специального зарядного устройства (ЗУ), подключаемого и отключаемого от сети с помощью коммутирующего устройства (КУ). Процессы включения и отключения зарядного устройства от сети, а также управление емкостным накопителем энергии выполняются и контролируются системой управления (СУ). Разряд емкостного накопителя энергии на индуктор с сопротивлением R и индуктивностью L производится после подачи отпирающего импульса на управляющий вентиль, работающий либо в ручном, либо в автоматическом режиме. В качестве индуктора здесь используется многовитковый соленоид.

Для полного уничтожения следов остаточной намагниченности носитель необходимо намагнитить до насыщения, а затем постепенно снизить напряженность поля до нуля, что и происходит, когда индуктор работает в колебательном режиме переходного процесса.

Направление внешнего магнитного поля задается конструкцией и формой витков индуктора. Для обеспечения максимальной эффективности намагничивания внешнее поле должно прикладываться в той же плоскости, в которой работают головки записи НЖМД.

Магнитное поле, генерируемое намагничивающими установками при достаточной амплитуде намагничивающего импульса, приводит к уничтожению служебной разметки поверхности диска и данных в секторах. При этом НЖМД выходит из строя, так как механика привода не может функционировать без служебной разметки диска, что приводит к невозможности проверки надежности уничтожения информации. Убедиться в том, что информация

Таблица 12.2

Характеристики системы «РАСКАТ-УЛЬТРА»

Параметр	Значение
Число одновременно стираемых носителей	В зависимости от типа устройства от 1 до 8 винчестеров в одной промышленной стойке
Напряжение питания	220 В 12 В
Напряженность стирающего магнитного поля	Не менее 700 кА/м
Время готовности к стиранию информации после включения питания	10 с (при питании от сети 220 В)
Длительность стирания информации	Не более 0,1 с
Время работы в автономном режиме	Не менее 24 ч
Время нахождения в дежурном режиме (готовности к стиранию)	Круглосуточно
Дальность действия радиоканала	До 50 м (в условиях прямой видимости), при усилении — до 1 км (в условиях прямой видимости)

уничтожена, позволяют только средства визуализации магнитных полей носителя.

Общие характеристики одной из систем уничтожения данных «РАСКАТ-УЛЬТРА» приведены в табл. 12.2.

12.4. Программа для создания зашифрованной области на жестком диске DriveCrypt Plus Pack 3

Рассмотрим программный комплекс DriveCrypt Plus Pack 3 компании SecurStar GmbH, который используют многие российские предприятия для защиты конфиденциальной информации.

Надежное хранение конфиденциальных данных обеспечивается при выполнении следующих условий:

• компьютер, на котором хранится конфиденциальная информация, не должен быть подключен ни к каким сетям (локальным, глобальным);

• зашифровывать следует не только данные (создание контейнера), но и все содержимое жесткого диска;

• используемая система криптозащиты должна уметь создавать ложную операционную систему, доступ к которой возможен только при вводе пользователем определенного пароля;

• экран для ввода пароля не должен раскрывать факт использования системы криптозащиты (в идеале он должен маскироваться под системное сообщение);

• попытки подбора паролей по словарю должны пресекаться системой криптозащиты;

• переустановка операционной системы, подключение винчестера к другой машине, работа в ложной операционной системе не должны выдавать факта использования системы криптозащиты;

• продолжительная работа в ложной системе должна вызывать частичное уничтожение или разрушение структуры зашифрованных данных;

• работа с криптосистемой должна быть максимально прозрачна для пользователя, но не должна требовать от него каких-либо специальных знаний или выполнения каких-либо действий с определенной периодичностью.

Комплекс DriveCrypt Plus Pack 3 отвечает всем указанным условиям.

Возможности этого программного комплекса в процессе шифрования жесткого диска и создание ложной операционной системы рассмотрим с помощью схемы, приведенной на рис. 12.2.

Работа программы по шифрованию включает в себя выполнение следующих этапов:

• создание и регистрация ключей электронной цифровой подписи (ЭЦП);

• защита и ограничение доступа к операционной системе Windows;

• шифрование загрузочного диска или раздела;

• шифрование разделов, дисков или сменных носителей;

• создание ложной операционной системы (ОС).

Установка DriveCrypt Plus Pack 3 выполняется мастером и не требует каких-либо специальных знаний. После запуска инсталлятора на экран поочередно выводятся стандартные окна мастера установки, в которых надо принять условия лицензионного соглашения, выбрать папку для установки программы, указать имя исполняемого файла, а также указать, где должны быть созданы ярлыки для запуска комплекса. На этом же экране мастера установки можно выбрать включение автоматического пароля при загрузке ОС. На следующем экране можно отменить установку справочной системы и ассоциацию файлов некоторого типа. На заключительном этапе установки необходимо перезагрузить компьютер.

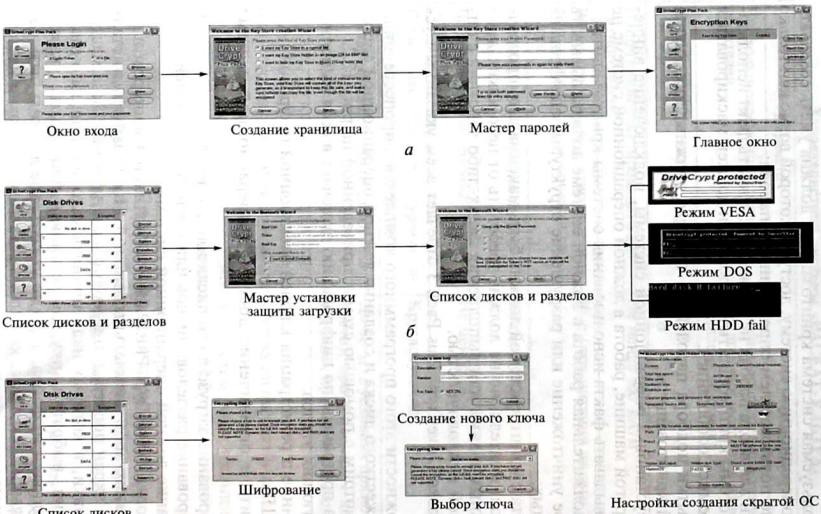


Рис. 12.2. Этапы работы программы по шифрованию носителя:

а — процесс создания и регистрации ключей ЭЦП; *б* — процесс защиты и ограничения доступа к среде Windows; *в* — шифрование загрузочного диска или раздела; *г* — шифрование разделов, дисков, сменных носителей; *д* — создание ложной операционной системы

После первого запуска DriveCrypt Plus Pack 3 предлагается либо зарегистрировать программу, либо использовать ее в ознакомительных целях.

Затем на экран будет выведено *Окно входа*, в котором следует нажать кнопку [Create] для создания хранилища ключей, после чего откроется окно мастера *Создание хранилища*. При последующих сеансах работы с программой в данном окне необходимо выбрать хранилище и ввести пароли, после чего будет предоставлен доступ к настройкам программы. В этом же окне выбирают тип файла, в котором будет размещено хранилище (текстовый, графический, аудио- или видеофайл).

На следующем экране выбирают папку, где будет размещено хранилище, и указывают имя файла. Если на предыдущем экране было выбрано создание хранилища в графическом или аудиофайле, то на этом экране следует указать существующий файл.

После указания имени и пути к хранилищу вводят пароли с помощью мастера паролей.

Существует два типа паролей: *Мастер* и *Пользователь*. Отличаются они доступом к настройкам программы. Пароль *Пользователь* не дает возможности что-то изменить, он позволяет лишь просматривать заданные мастером настройки. Каждый тип пароля, в свою очередь, также может состоять из двух паролей. Эту особенность можно использовать для создания пары паролей, каждый из которых будет известен только одному человеку. При создании паролей нельзя использовать шрифт, содержащий знаки кириллицы.

После создания хранилища на экран выводится *Главное окно*, в котором показаны ключи, находящиеся в текущем хранилище. Если хранилище только что создано, то ключей еще нет и их необходимо создать при помощи кнопки [New Key]. Ключи шифруются по определенному программой алгоритму (AES 256).

После создания ключей можно приступать к защите загрузки, для чего в главном окне программы следует нажать кнопку [Drives]. В результате откроется список дисков и разделов, пример которого показан на рис. 12.2 в окне *Список дисков и разделов*.

Выделив в этом списке загрузочный диск или раздел и нажав кнопку [Bootauth], запускают мастер установки защиты загрузки.

На следующем этапе установки защиты в окне *Список дисков и разделов* предлагается ограничить загрузку в зависимости от введенного пароля. Доступ к данным здесь предоставляется как по паролю *Мастер*, так и по паролю *Пользователь*. Если пароль *Пользователь* не был указан на этапе создания хранилища, то часть опций в этом окне будет недоступна. В этом случае загрузка компьютера разрешается только по паролю *Мастер*.

На заключительном этапе установки защиты выводится информационное сообщение о том, что перед шифровкой любых дис-

ков следует проверить корректность установки защиты загрузки. Тогда если что-то пойдет не так (например, будет забыт пароль *Мастер*), на этом этапе можно очень просто восстановить загрузку операционной системы. Если же зашифровать диск без проверки корректности работы защиты загрузки, восстановить его содержимое будет невозможно.

После перезагрузки компьютера до момента загрузки ОС на экран выводится приглашение к вводу одного из трех видов паролей: *Режим VESA*, *Режим DOS*, *Режим HDD fail*.

Затем вводят в поля соответствующие пароли. Для перехода от первого поля ко второму используют клавишу [Tab]. При выборе режима HDD fail вводимые пароли не отображаются символами. После введения в этом режиме первого пароля следует снова нажать клавишу [Tab], затем ввести второй пароль (если он был задан при создании хранилища) и нажать клавишу [Enter]. Для ввода пароля дается три попытки. Если все три попытки неудачны, система останавливается и для повторного ввода пароля следует перезагрузить компьютер.

После тестирования защиты загрузки можно переходить к шифрованию диска или раздела.

Шифрование загрузочного диска или раздела. Шифрование загрузочного диска или раздела невозможно без установки защиты загрузки.

Чтобы зашифровать диск или раздел, следует сделать соответствующий выбор в окне *Список дисков* (Disk Drives) и нажать кнопку [Encrypt]. При этом мастер шифрования диска откроет окно *Шифрование*, в котором будет предложено выбрать ключ из хранилища. Диск будет зашифрован этим ключом и этот же ключ потребуется для дальнейшей работы с диском.

Выбор ключа запускает процесс шифрования диска. Этот процесс достаточно длительный, он может занимать до нескольких часов в зависимости от объема шифруемого диска или раздела.

После завершения шифрования загрузочного диска загрузка операционной системы возможна только после проведения авторизации.

После того как загрузочный диск будет полностью зашифрован, мастер предложит создать аварийный диск. Отказываться от этого предложения крайне неразумно — аварийный диск лучше сразу создать и убрать в надежное место. Он поможет спасти данные в случае проблем с загрузкой операционной системы. Если аварийный диск не создать, то даже случайная перезапись главной загрузочной записи приведет к полной и безвозвратной потере всех данных, которые хранились на зашифрованном диске.

Шифрование разделов, дисков, сменных носителей. При помощи комплекса DriveCrypt Plus Pack 3 можно зашифровать любой жесткий диск или сменный накопитель (за исключением CD и

DVD) и использовать его для обмена данными между пользователями. При этом следует позаботиться о передаче через надежный канал ключа, которым будут шифроваться данные.

Если планируется обмен информацией на зашифрованных сменных носителях, лучше создать отдельный ключ, которым будет зашифрован носитель. Для этого надо открыть главное окно, авторизоваться и на вкладке *Encryption Keys* нажать кнопку [New Key], после чего откроется окно *Создание нового ключа*, в котором следует ввести понятное описание вновь создаваемого ключа и нажать кнопку [Generate].

После создания нового ключа надо подключить сменный носитель и переключиться на вкладку *Drives*. Если в списке отсутствует только что подключенный накопитель, следует обновить его при помощи кнопки [Refresh], выделить сменный накопитель и нажать кнопку [Encrypt], после чего откроется окно *Выбор ключа* для шифрования.

После выбора созданного ключа, предназначенного только для шифрования сменного носителя при обмене информацией, следует нажать кнопку [Encrypt], и сменный носитель будет зашифрован.

Теперь нужно экспортировать в файл ключ, которым был зашифрован сменный носитель. Для этого вновь надо переключиться на вкладку *Encryption Keys*, выделить только что созданный ключ и нажать кнопку [Export]. В открывшемся при этом окне выбирают папку, где следует сохранить ключ, имя файла с ключом и пароль для защиты самого ключа. Полученный файл надо передать пользователю, с которым планируется обмениваться информацией на зашифрованных носителях, а также сообщить ему пароль, которым был защищен ключ во время экспорта. Когда этот пользователь получит файл, он должен будет импортировать ключ в хранилище, после чего получит доступ к переданной информации на зашифрованном диске.

Импорт ключа выполняется по аналогии с экспортом с помощью соответствующей кнопки на вкладке *Encryption Keys*. После того как получатель зашифрованного диска импортировал ключ в хранилище, он может установить в привод или подключить к компьютеру сменный носитель и открыть его. Для этого пользователь должен переключиться на вкладку *Drives*, выделить в списке сменный накопитель и нажать кнопку [Explore]. При последующей работе с зашифрованным сменным накопителем пользователь должен авторизовываться после каждой перезагрузки программы, так как до тех пор пока он этого не сделает, сменный накопитель будет закрыт и система будет предлагать отформатировать его при каждом обращении.

Создание ложной операционной системы. Создать ложную операционную систему достаточно просто. Для этого следует создать

раздел и отформатировать его в FAT32, установить и настроить Windows и установить DriveCrypt Plus Pack 3. Установленную операционную систему надо полностью настроить, скопировать какие-то файлы, например на рабочий стол, установить какое-то программное обеспечение и т.д. Создаваемая ложная операционная система должна выглядеть как рабочая, т.е. постоянно используемая. После создания скрытой операционной системы загружаться и работать с ложной ОС крайне опасно, поскольку существует вероятность разрушения данных скрытой операционной системы.

После установки и настройки ложной операционной системы следует создать новое хранилище, авторизоваться в DCPP, переключиться на вкладку *Drives*, выделить раздел, где установлена ложная операционная система (файловая система должна быть обязательно FAT32, иначе создать ложную ОС будет невозможно), и нажать кнопку [HiddenOS]. В результате откроется окно *Настройка создания скрытой ОС*.

Затем надо указать путь к только что созданному хранилищу, пароли, метку скрытого диска, его файловую систему и размер свободного места, которое должно отделять ложную операционную систему от скрытой. Нажатием кнопки [Create Hidden OS] запускается процесс создания скрытого раздела и все содержимое системного раздела копируется в скрытый раздел.

DriveCrypt Plus Pack 3 создает скрытый раздел, начало которого находится через определенный промежуток свободного места от окончания ложного раздела, указанный при создании скрытого раздела. Если при этом загрузиться и начать работу в ложной ОС, то после того как указанное при создании скрытого раздела свободное место будет использовано, ложная операционная система начнет перезаписывать зашифрованные данные скрытой ОС, что приведет к ее неработоспособности.

После завершения процесса создания скрытого раздела следует перезагрузить компьютер и авторизоваться, введя пароли, которые были указаны при создании скрытого раздела. Содержимое ложной операционной системы не будет видно при работе в скрытой ОС, и наоборот: при работе в ложной операционной системе не будет видно содержимое скрытой ОС. Таким образом, только введенный пароль при включении компьютера определяет, какая операционная система будет загружена.

После окончания создания скрытой операционной системы в нее следует войти и зашифровать системный раздел.

Тестирование производительности дисковой подсистемы. Для определения того, насколько комплекс DriveCrypt Plus Pack 3 снижает производительность операций считывания-записи, тестируют скорость работы дисковой подсистемы. Результаты такого тестирования приведены в табл. 12.3.

Таблица 12.3

Результаты тестирования скорости работы дисковой подсистемы

Операция	Скорость до шифрования, Мбайт/с	Скорость после шифрования, Мбайт/с	Падение скорости, %
Буферизованное считывание	89	16	82
Последовательное считывание	34	13	62
Случайное считывание	24	13	46
Буферизованная запись	56	15	73
Последовательная запись	33	15	55
Случайная запись	20	13	35

Результаты тестирования показывают, что падение скорости операций считывания-записи весьма существенное. Если на компьютере планируется работа с программным обеспечением, активно использующим жесткий диск, установка комплекса DCPP значительно снизит скорость работы.

На схеме, представленной на рис. 12.3, показан процесс работы программы по созданию дисков с зашифрованной областью данных, которые размещаются в дисковом хранилище, связанном с устройством уничтожения данных.

Диск аварийного восстановления. Когда установлена защита загрузки и зашифрован системный раздел, комплекс DriveCrypt Plus Pack 3 автоматически предлагает создать аварийный диск, при помощи которого можно восстановить работу компьютера в случае возникновения внештатных ситуаций. Аварийный диск создается с помощью мастера, и его создание не вызывает каких-либо затруднений.

Аварийный диск поможет авторизоваться в DCPP при проблемах с загрузкой с жесткого диска, а также расшифровать зашифрованный диск в обычной или скрытой ОС.

Таким образом применение программы DriveCrypt Plus Pack 3 обеспечивает пользователю уверенность в том, что его данные не будут прочитаны даже в случае хищения компьютера или кражи сменного носителя. Возможность создания ложной системы, при работе в которой будут уничтожаться данные основной системы, поможет ввести дополнительный уровень защиты. Способность программы DriveCrypt Plus Pack 3 шифровать целый диск или раздел позволяет скрыть не только важные данные, но и все содержимое диска или раздела, включая операционную систему.

Рис. 12.3. Схема работы программы по созданию дисков с зашифрованной областью данных

К сожалению, за такой уровень безопасности приходится расплачиваться значительным падением производительности файловой системы, что в некоторых случаях может стать серьезным препятствием в использовании DriveCrypt Plus Pack 3.

12.5. Организационные рекомендации по обеспечению безопасности эксплуатации удаленных баз данных

Ранее были рассмотрены программные и технические методы защиты информации. Однако все эти средства защиты будут эффективно функционировать только при соблюдении жестких контролируемых организационных мер. Приведем перечень необходимых организационных мероприятий.

1. Требования к организации помещений с компьютерным оборудованием.

1.1. Размещение специального оборудования должно обеспечивать невозможность неконтролируемого проникновения в эти помещения посторонних лиц.

1.2. Помещения с компьютерным оборудованием должны находиться в контролируемой зоне, иметь прочные входные двери с надежными замками и средствами регистрации доступа. При расположении этих помещений на первых и последних этажах зданий, а также при наличии рядом с их окнами балконов и пожарных лестниц рекомендуется оборудовать окна внутренними (раздвижными) решетками. Также двери и окна данных помещений рекомендуется оборудовать охранной сигнализацией.

2. Требования по организации хранения и использования ключевой информации.

2.1. Порядок хранения и использования ключевых дискет с секретными ключами подписи и шифрования должен исключать возможность несанкционированного доступа к ним.

2.2. К каждой ключевой дискете должен иметь доступ только один человек — владелец записанных на ней ключей, который несет персональную ответственность за их использование.

2.3. Для хранения секретных ключей в помещениях с компьютерным оборудованием должны иметься надежные металлические хранилища (сейфы), оборудованные надежными запирающими устройствами с двумя экземплярами ключей (один ключ должен быть у исполнителя, другой — в службе безопасности).

2.4. По окончании рабочего дня носители ключевой информации следует убирать в сейф.

2.5. Во время работы с секретными ключами должен быть исключен доступ к носителям ключевой информации кого-либо, кроме их владельца.

2.6. Хранение секретных ключей допускается в одном хранилище с другими документами, но отдельно от них и в упаковке, исключающей возможность негласного доступа к ним.

2.7. Для обеспечения работоспособности системы в случае сбоя носителя ключевой информации должно быть организовано резервное копирование ключевых дискет. При этом хранение резервных копий ключевых дискет и доступ к ним должны быть организованы в соответствии со следующими требованиями:

- не допускается снятие несанкционированных копий с ключевых носителей;
- не допускается знакомить кого-либо с содержанием ключевых носителей или передавать кому-либо ключевые носители;
- не допускается вывод секретных ключей подписи и шифрования на дисплей или принтер;
- нельзя вставлять ключевую дискету в дисковод компьютера в режимах, не предусмотренных функционированием системы, а также в дисководы других компьютеров;
- нельзя записывать на дискету с ключами постороннюю информацию.

3. Требования к защите программного обеспечения комплекса и ключевой информации от несанкционированного доступа.

Меры, перечисленные в данном пункте, принимаются в целях исключения возможности:

- появления в компьютерах компьютерных вирусов и программ, направленных на разрушение или модификацию СУБД либо на перехват паролей и секретных ключей;
- внесения несанкционированных изменений в технические и программные средства СУБД, а также в их состав.

3.1. Программное обеспечение СУБД для удаленного доступа к данным следует устанавливать на отдельный специально выделенный для этих целей компьютер (сервер БД) с «чистой» операционной средой.

3.2. Не допускается установка на сервер БД программных средств, не предназначенных для решения задач по работе с базами данных.

3.3. В целях защиты СУБД от несанкционированного доступа (НСД) на компьютер следует устанавливать программно-аппаратный комплекс Secret Net или аналогичный комплекс, сертифицированный Госстандартом России по третьему классу.

3.4. В соответствии с планом защиты следует формировать с помощью комплекса защиты от НСД функционально замкнутую среду, допускающую работу пользователей строго в рамках возможностей, предоставляемых им программным обеспечением абонентского пункта, и полномочий, установленных согласно принятой на предприятии политики обеспечения безопасности.

3.5. Следует обеспечивать непрерывный контроль целостности программного обеспечения системы.

3.6. Следует принять меры, препятствующие извлечению платы Secret Net Card (для автономного варианта) или ROM BIOS (для сетевого варианта) комплекса Secret Net либо аналогичных аппаратных средств защиты от НСД, а также опечатать системный блок компьютера и обеспечить периодический контроль целостности печатей.

3.7. Порядок хранения и использования персональных дискет должен исключать возможность их несанкционированного использования.

3.8. Права супервизора комплекса Secret Net либо аналогичных средств защиты от НСД следует предоставить только администратору безопасности (защиты от НСД).

4. Требования к персоналу управления базами данных.

4.1. Список пользователей, допускаемых к работе по управлению СУБД, утверждается приказом по предприятию с закреплением за каждым пользователем конкретных функций и полномочий.

4.2. К работе допускаются пользователи, в совершенстве знающие правила обеспечения безопасности и владеющие практическими навыками работы на компьютере.

4.3. Пользователи допускаются к работе только после изучения и проверки знаний правил эксплуатации.

4.4. Пользователи дают расписки о неразглашении и нераспространении конфиденциальной информации, секретных ключей и паролей.

4.5. Лица, работающие с управлением базами данных, подразделяются на следующие категории:

- администратор безопасности (защиты от НСД), в функции которого входят регистрация пользователей, допущенных к работе и контроль за выполнением требований безопасности;
- администратор, осуществляющий подготовку электронных документов и выработку электронной цифровой подписи;
- уполномоченный администратор, дополнительно осуществляющий взаимодействие с пунктом регистрации клиентов;
- клиент, ответственный за прием и передачу электронных документов.

4.6. При необходимости решения текущих и оперативных вопросов в части безопасности администратор БД и администратор защиты от НСД взаимодействуют со службой безопасности предприятия.

Контрольные вопросы

1. Что означает понятие *защита информации, хранящейся в базах данных*?
2. Какие потенциальные опасности существуют при эксплуатации баз данных?

3. Назовите возможные угрозы безопасности информационных систем.

4. Какие факторы определяют технологическую безопасность информационных систем?

5. В чем состоит различие понятий *характеристика степени безопасности* и *показатели надежности* информационных систем?

6. Поясните следующие критерии безопасности: устойчивость, восстанавливаемость, коэффициент готовности.

7. Назовите методы обеспечения технологической безопасности информационных систем.

8. Что означает понятие *авторизация пользователей*?

9. Назовите привилегии языка SQL, которые устанавливает стандарт ISO/EC 9075:2003, и укажите их назначение.

10. Для чего применяются RAID-технологии?

11. Какие методы физического изменения структуры магнитного материала рабочих поверхностей жестких дисков вы знаете?

12. Для чего применяется программа DriveCrypt Plus Pack 3?

13. Какие части содержит диск с зашифрованной областью данных?

14. Каковы требования к организации помещений с компьютерным оборудованием управления удаленными базами данных?

15. Каковы требования к организации хранения и использования ключевой информации?

16. Назовите требования, предъявляемые к персоналу управления базами данных.

ГЛАВА 13

ВОССТАНОВЛЕНИЕ ДАННЫХ В КРИТИЧЕСКИХ СИТУАЦИЯХ

13.1. Восстановление базы данных

Восстановление базы данных — это процесс возвращения базы данных в рабочее состояние, утраченное в результате сбоя или отказа.

Для хранения данных используют четыре различных типа носителей, которые приведем в порядке возрастания их надежности:

- *оперативная память*, представляющая собой временное хранилище информации, содержимое которого в случае отказа системы обычно разрушается;
- *магнитные диски*, представляющие собой оперативное постоянное хранилище информации. Диски более надежны и значительно дешевле, чем оперативная память, однако скорость доступа к информации у них меньше на три-четыре порядка;
- *магнитная лента*, представляющая собой автономный постоянный носитель информации, надежность которого существенно выше, чем у магнитного диска, а стоимость намного ниже. Однако этот носитель предоставляет только последовательный доступ к информации, причем с относительно небольшой скоростью;
- *оптические диски*, являющиеся более надежными, чем магнитные ленты, и при этом достаточно недорогими, более быстродействующими и к тому же допускающими произвольный доступ к информации.

Оперативную память часто называют *первичной* памятью, а магнитные оптические диски и магнитные ленты — *вторичной*, или *внешней*, памятью.

Устойчивые хранилища обеспечивают надежное сохранение информации, размещая несколько ее копий на различных постоянных носителях (обычно на дисках), одновременный отказ которых маловероятен. В частности, устойчивое хранение информации может быть организовано с помощью RAID-технологии, гарантирующей, что отказ отдельного дискового устройства (даже в процессе передачи данных) не вызовет потери данных.

Существует множество различных типов отказов, способных повлиять на функционирование базы данных, каждый из которых требует особых способов устранения. Одни отказы влияют только на содержимое оперативной памяти, другие — могут воздействовать и на постоянную (вторичную) память системы.

Приведем некоторые причины, способные вызвать отказы в работе устройств хранения информации:

- *аварийное прекращение работы* системы, вызванное ошибкой оборудования или программного обеспечения, приведшее к разрушению содержимого оперативной памяти;
- *отказ носителей информации*, вызванный разрушением устройств считывания, что может привести к потере части содержимого вторичной памяти системы;
- *ошибки прикладных программ* (например, логические ошибки в программах, получающих доступ к базе данных), послужившие причиной сбоев при выполнении одной или нескольких транзакций;
- *стихийные бедствия* — пожары, наводнения, землетрясения или отказы в электропитании, приводящие к разрушению носителей информации;
- *небрежное обращение* со стороны операторов или пользователей системы, которое является причиной непреднамеренного разрушения данных или программ;
- диверсии, или преднамеренное разрушение и уничтожение данных, оборудования или программного обеспечения.

Однако какой бы ни была причина отказа системы, существуют два принципиальных его последствия, которые следует учитывать:

- утрата содержимого оперативной памяти;
- утрата копии базы данных на дисках.

Далее рассмотрим на концептуальном уровне технологии, позволяющие минимизировать последствия аварий и успешно восстановить систему после сбоя.

13.2. Транзакции и восстановление

Транзакции представляют собой основную единицу восстановления в системах управления базами данных.

Диспетчер восстановления СУБД должен обеспечивать поддержку следующих основных свойств транзакций: неразрывности и устойчивости при наличии сбоев в системе. Также он должен обеспечить, чтобы при восстановлении после сбоя для каждой отдельной транзакции в базе данных либо постоянно фиксировались все внесенные ею изменения, либо не фиксировалось ни одно из них.

Ситуация осложняется тем фактом, что запись в базу данных не представляет собой неразрывное действие (выполняемое за один шаг). Следовательно, существует вероятность того что, когда выполнение транзакции будет завершено посредством фиксации, внесенные ею изменения не будут реально отражены в базе

данных по той простой причине, что они еще не достигли ее файлов.

При выполнении операции считывания информации СУБД осуществляет следующие типовые действия:

- определяет дисковый адрес блока данных, содержащего запись с первичным ключом *x*;
- считывает блок данных с диска и помещает его в буфер СУБД, находящийся в оперативной памяти;
- копирует данные из буфера СУБД и помещает их в динамическую таблицу.

При выполнении операции записи информации СУБД осуществляет следующие действия:

- определяет дисковый адрес блока данных, содержащего запись с первичным ключом *x*;
- считывает блок данных с диска и помещает его в буфер СУБД, находящийся в оперативной памяти;
- копирует данные из динамической таблицы и помещает их в буфер СУБД;
- выводит блок данных из буфера в оперативной памяти и помещает их на диск.

Буферы СУБД занимают определенную часть оперативной памяти и используются для обмена данными со вторичной памятью системы.

Только после выгрузки соответствующего буфера во вторичную память можно считать, что выполненные операции обновления приобрели постоянный характер.

Выгрузка буферов в базу данных может осуществляться по специальной команде (например, по команде фиксации транзакции) или же автоматически, как только буфер будет заполнен.

Выдачу явного указания о необходимости записи содержимого буферов во вторичную память называют принудительной записью.

Если отказ системы произойдет между записью данных в буфер и выгрузкой буфера во вторичную память, диспетчер восстановления должен уточнить состояние транзакции, выполнявшей запись в момент аварии.

Если транзакция уже выдала команду фиксации, то для обеспечения устойчивости ее результатов диспетчер восстановления должен выполнить ее повторно (redo), чтобы восстановить все внесенные изменения. Эту операцию часто называют *накатом*.

С другой стороны, если на момент отказа системы транзакция еще не была зафиксирована, диспетчер восстановления должен отменить (undo) любые ее результаты, т. е. выполнить их откат.

Выполнение отката только одной транзакции называется *частичным откатом*.

Выполнение отката всех активных транзакций называется *глобальным откатом*.

13.3. Управление буферами базы данных

Организация управления буферами базы данных играет важную роль в процессе восстановления информации. Рассмотрим применяемые для этого методы.

Процессы управления буферами базы данных, которые применяются для считывания и записи страницы данных во вторичную память, осуществляются специальной программой, называемой диспетчером буферов. Такая программа должна предусматривать считывание страниц с диска в буферы до полного их заполнения, а затем применение той или иной стратегии замещения для определения того, какой буфер (буферы) необходимо принудительно записать на диск, чтобы освободить место для новых страниц, которые должны быть считаны с диска.

Один из алгоритмов управления предусматривает применение двух буферов переменных — pinCount и dirty. Этим переменным, соответствующим каждому буферу базы данных, первоначально присваивается нулевое значение.

При получении запроса на считывание страницы с диска диспетчер буферов выполняет проверку для определения, не находится ли эта страница в одном из буферов.

Если страница не находится ни в одном из буферов, то диспетчер буферов выполняет следующие действия:

- выбирает буфер, предназначенный для замещения (называемый *замещаемым буфером*), и увеличивает значение переменной на единицу;
- если переменная dirty равна нулю, производит запись буфера на диск;
- считывает страницу с диска в буфер замещения и присваивает переменной dirty нулевое значение.

При поступлении повторного запроса к одной и той же странице соответствующее значение pinCount увеличивается на единицу. После того как система сообщит диспетчеру буферов, что применение этой страницы закончено, соответствующее значение pinCount уменьшается на единицу. Кроме того, система может сообщить диспетчеру буферов, что в содержимое страницы внесено изменение, поэтому переменной dirty присваивается ненулевое значение, т.е. страница отмечается как «грязная» — требующая записи на диск. После того как значение pinCount станет нулевым, страница становится незакрепленной и может быть записана на диск.

Приведем правила восстановления базы данных при записи страниц на диск:

- *правило конфискации* позволяет диспетчеру буферов записывать содержимое буфера на диск прежде, чем транзакция выполнит фиксацию (буфер при этом не закреплен). Иными словами,

диспетчер буферов может «отнять» страницу, принадлежащую транзакции. Правило, противоположное данному, запрещает конфискацию страниц;

• *правило принудительной записи* гарантирует, что все страницы, модифицированные в ходе транзакции, будут немедленно записаны на диск после фиксации транзакции. Правило, противоположное данному, не требует принудительной записи страниц.

На практике проще всего реализовать подход к восстановлению БД, предусматривающий использование правила, запрещающего конфискацию страниц, и правила принудительной записи.

При использовании правила, запрещающего конфискацию страниц, не требуется выполнения отката транзакции, завершившейся аварийно, поскольку в этом случае внесенные изменения еще не были записаны на диск. При этом применение правила принудительной записи гарантирует, что при возникновении аварии в системе не придется выполнять накат изменений, внесенных зафиксированной транзакцией, так как все изменения после фиксации транзакции немедленно записываются на диск.

Здесь правило, запрещающее конфискацию страниц, используется в протоколе отложенного восстановления результатов обновления.

С другой стороны, правило, допускающее конфискацию страниц, имеет определенное преимущество перед правилом, не допускающим такую конфискацию. Данное преимущество заключается в том, что позволяет избежать необходимости распределения объема буферного пространства, который может потребоваться для хранения всех копий страниц обновляемых многочисленными параллельными транзакциями.

Типовая структура СУБД должна предоставлять следующие функции восстановления:

• механизм резервного копирования, предназначенный для периодического создания резервных копий базы данных;

• средства ведения журнала, в котором фиксируются текущее состояние транзакций и вносимые в базу данных изменения;

• фиксацию создания контрольных точек, обеспечивающую перенос выполняемых в базе данных изменений во вторичную память с целью сделать их постоянными;

• диспетчера восстановления, обеспечивающего восстановление согласованного состояния базы данных, нарушенного в результате отказа.

13.4. Механизм резервного копирования

Любая СУБД должна предоставлять механизм, позволяющий создавать резервные копии базы данных и файла ее журнала через установленные интервалы времени.

Резервное копирование может выполняться как для всей базы данных в целом, так и для изменившейся ее части. В последнем случае в копию помещаются сведения только об изменениях, накопившихся с момента создания предыдущей полной или измененной резервной копии системы. Как правило, резервные копии создаются на автономных носителях.

Файл журнала. Для фиксации хода выполнения транзакций СУБД использует специальный файл, который называют *журналом*. Этот файл содержит сведения обо всех обновлениях, выполненных в базе данных. В файл журнала помещают записи о транзакциях и записи контрольных точек.

Записи о транзакциях включают в себя:

- идентификатор транзакции;
- тип записи журнала (начало транзакции, операции вставки, обновления или удаления, отмену или фиксацию транзакции);
- идентификатор элемента данных, вовлеченного в операцию обработки базы данных (операции вставки, удаления и обновления);
- копию элемента данных до операции, т.е. его значение до изменения (только операции обновления и удаления);
- копию элемента данных после операции, т.е. его значение после изменения (только для операций обновления и вставки);
- служебную информацию файла журнала, содержащую указатели на предыдущую и следующую записи журнала для этой транзакции (все операции).

Создание контрольных точек. Помещаемая в файл журнала информация предназначена для использования в процессе восстановления системы после отказа. Однако при возникновении отказа может отсутствовать информация о том, с какого прошедшего момента необходимо начинать поиск в файле журнала, чтобы не выполнить накат транзакций, которые завершились с успешной фиксацией в базе данных.

Для ограничения объема поиска и последовательной обработки информации в файле журнала используют метод создания контрольных точек.

Контрольная точка — момент синхронизации между базой данных и журналом регистрации транзакций, при котором все буферы системы принудительно записываются в ее вторичную память.

Контрольные точки создаются через установленный интервал времени и предусматривают выполнение следующих действий:

- перенос всех имеющихся в оперативной памяти записей журнала во вторичную память;
- запись всех модифицированных блоков в буферах базы данных во вторичную память;
- помещение в файл журнала записи контрольной точки, которая содержит идентификаторы всех транзакций, активных в момент ее создания.

Если транзакции выполняются последовательно, то после возникновения отказа файл журнала просматривается в целях обнаружения последней транзакции, выполненной до создания последней контрольной точки.

Если транзакции выполняются параллельно, требуется выполнение наката всех транзакций, которые были зафиксированы со времени создания контрольной точки, и отката всех транзакций, которые были активны в момент аварии.

Контрольные вопросы

1. В каких случаях производят восстановление базы данных?
2. Какие устройства называются первичной памятью и какие вторичной, или внешней?
3. Какие причины способны вызвать отказы в работе устройств хранения информации?
4. Что является основной единицей восстановления в системах управления базами данных?
5. Какие задачи решает диспетчер восстановления СУБД?
6. Какие операции называются накатом и откатом?
7. Какие операции называются частичным и глобальным откатом?
8. Что представляет собой буфер базы данных и каковы процессы управления буферами базы данных?
9. Как называется файл для фиксации хода выполнения транзакций и какие сведения он должен содержать?
10. Для чего служит контрольная точка?

ЧАСТЬ VI

ПОСТРЕЛЯЦИОННЫЕ СИСТЕМЫ УПРАВЛЕНИЯ УДАЛЕННЫМИ БАЗАМИ ДАННЫХ

ГЛАВА 14

ОРИЕНТАЦИЯ РАЗВИТИЯ СУБД НА РАСШИРЕННУЮ РЕЛЯЦИОННУЮ МОДЕЛЬ

14.1. Основные направления совершенствования реляционных баз данных

Рассмотрим основные направления в области исследований и разработок систем управления так называемых постреляционных баз данных.

Можно отметить три направления развития СУБД следующего поколения.

Первое направление — максимальное использование существующих технологий управления и дальнейшее совершенствование систем управления внешней памятью реляционных СУБД.

Второе направление — создание генераторов систем управления в виде наборов модулей со стандартизованными интерфейсами.

Третье направление развития СУБД по сути является синтезом первых двух направлений. СУБД проектируется как некоторый интерпретатор системы правил и набор модулей — действий, вызываемых в соответствии с этими правилами. Для таких систем разрабатываются специальные языки формирования правил.

Можно сказать, что СУБД следующего поколения — это прямые наследники реляционных систем. Тем не менее различные направления развития систем третьего поколения стоит рассмотреть отдельно, поскольку они обладают разными характеристиками.

Одним из основных положений реляционной модели данных является требование нормализации отношений. Это позволяет проектировать БД с предельно понятной структурой и экономить значительные ресурсы памяти компьютера. В этом случае для обеспечения целостности информации используются соответствующие связи между таблицами.

Однако реляционные СУБД стали применять не только в сфере бизнеса для решения управленческих задач, но и в сфере промышленного производства (CALS-технологии). Применение реляционных баз данных оказалось эффективным для разработки систем автоматизированного проектирования технологических процессов

(САПРТП), экспертных систем, а также для решения других задачах управления и технической подготовки производства.

Такие системы обычно оперируют сложноструктурированными объектами — некоторым комплексом логически связанных таблиц, для анализа информации которых, а тем более для выбора рациональных технических решений приходится выполнять сложные запросы.

В соответствии с такими задачами применения реляционных БД появилось новое направление их развития. Суть этого направления сводится к тому, что в системах управления базами данных формируются сложные объекты, объединяющие в себе не только исходные таблицы БД, но и соответствующие запросы. При этом сохраняется четкая граница между логическим и физическим представлениями таких объектов. В частности, для любого сложного объекта (произвольной сложности) обеспечивается возможность перемещения или копирования его как единого целого из одной части базы данных в другую ее часть или даже в другую базу данных.

Это очень обширная область исследований, в которой затрагиваются вопросы разработки моделей данных, структур данных, языков запросов, управления транзакциями, журнализации и т.д.

Новое направление в разработке СУБД хотя и основывается на реляционной модели, но в ней не обязательно поддерживается требование полной нормализации отношений.

С расширением области применения реляционного подхода для решения задач, особенно в направлении CALS-технологий, стало очевидным, что применение принципа нормализации таблиц БД и создание отдельных транзакций и процедур при выполнении различных запросов «сводит на нет» все преимущества нормализованной схемы организации базы данных.

В ненормализованных реляционных моделях данных допускается создание и хранение в качестве объекта (исходного элемента системы) кортежей (записей), массивов (регулярных индексированных множеств данных), регулярных множеств элементарных данных, а также отношений.

Системы управления базами данных, в которых формируются такие сложные объекты, называют *объектно-ориентированными базами данных* (ООБД).

Очевидно, что эти системы должны обязательно иметь в своем составе языки программирования для создания и управления такими объектами. Кроме того, они должны обрабатывать различные типы данных, в том числе формируемые пользователями.

В 1995 г. компания Sun Microsystems объявила о выпуске нового продукта — языка из семейства интерпретаторов под названием Java. Язык Java является расширенным подмножеством языка Си++, основное отличие которого состоит в том, что он является пооператорно интерпретируемым (в стиле языка Basic), а программы,

написанные на языке Java, гарантированно безопасны (в частности, при выполнении любой программы не может быть поврежден интерпретатор). Для этого из процедур языка удалены математические действия над указателями. В то же время Java остается мощным объектно-ориентированным языком, включающим в себя развитые средства определения абстрактных типов данных. Компания Sun Microsystems продвигает язык Java в целях расширения возможностей сети Интернет. Основная идея состоит в том, что с Web-сервера клиентам передаются не данные и не результаты обработки данных, а объекты, методы выполнения которых запрограммированы на языке Java и выполняются на стороне клиента.

14.2. Генерация систем баз данных, ориентированных на приложения

Появление данного направления в развитии СУБД определяется тем, что невозможно создать универсальную систему управления базами данных, которая будет достаточна и не избыточна для применения в любом приложении. Например, если посмотреть на использование существующих СУБД для решения практических задач в производстве и бизнесе, то можно утверждать, что в большинстве случаев применяется не более 30 % возможностей системы. Тем не менее приложение несет большую часть информационной нагрузки поддерживающей его СУБД, рассчитанной на использование в наиболее общих случаях.

Следовательно, очень заманчиво производить не законченные универсальные СУБД, а нечто вроде компиляторов (compiler compiler), позволяющих собрать систему баз данных, ориентированную на конкретное приложение (или класс приложений). Рассмотрим простые примеры.

Так, в системах резервирования проездных билетов запросы обычно настолько просты (например: «выдать очередное место на рейс № 645»), что нет смысла производить широкомасштабную оптимизацию запросов. Однако информация, хранящаяся в базе данных, настолько критична (кто из нас не сталкивался с проблемой наличия двух или более билетов на одно место?), что особенно важны гарантированные синхронизация обновлений базы данных и ее восстановление после любого сбоя.

В статистических системах запросы могут быть произвольно сложными (например: «выдать число холостых мужчин, проживающих в России и имеющих не менее трех зарегистрированных детей»), что вызывает необходимость использования развитых средств их оптимизации. Однако поскольку речь идет о статистике, здесь не требуется поддержка строгой сериализации транзакций и точного восстановления базы данных после сбоев. (Когда

речь идет о статистической информации, потеря нескольких ее единиц обычно не существенна.)

Из сказанного следует, что желательно уметь генерировать систему баз данных, возможности которой в достаточной степени соответствуют потребностям приложения. На сегодняшний день на коммерческом рынке такие генерационные системы отсутствуют (например, при выборе сервера системы Oracle при разработке конкретного приложения нельзя отказаться от каких-либо свойств системы).

14.3. Оптимизация запросов, управляемых правилами

Под оптимизацией запросов в реляционных СУБД обычно подразумевают такой способ их обработки, при котором по начальному представлению запроса в результате преобразований вырабатывается оптимальный процедурный план его выполнения.

Соответствующие преобразования начального представления запроса выполняются специальным компонентом СУБД — оптимизатором, и оптимальность производимого им плана выполнения запроса носит субъективный характер, поскольку критерий оптимальности заложен в него разработчиком.

Основная неприятность, связанная с оптимизаторами запросов, заключается в том, что отсутствует принятая технология их программирования. Обычно оптимизатор представляет собой некоторый набор относительно независимых процедур, которые жестко связаны с другими компонентами компилятора. По этой причине очень трудно менять стратегии оптимизации или качественно их расширять (делать это приходится, поскольку оптимизация вообще и оптимизация запросов в частности отражает эмпирические зависимости, а хорошие эмпирические алгоритмы появляются только со временем).

Решают эту проблему с помощью компромиссных решений, не выходящих за пределы традиционной технологии производства компиляторов. В основном все они связаны с применением тех или иных инструментальных средств, обеспечивающих автоматизацию построения компиляторов. Такие инструментальные средства имеются, например, в системах DB2, Oracle, Informix.

14.4. Поддержка динамической информации и темпоральных запросов

Обычные реляционные БД хранят мгновенный снимок модели предметной области. Любое изменение в момент времени *t* неко-

торого объекта приводит к недоступности состояния этого объекта в предыдущий момент времени. На самом деле в большинстве развитых СУБД предыдущее состояние объекта сохраняется в журнале изменений, но возможности доступа к нему со стороны пользователя нет.

Конечно, можно ввести в хранимые отношения временной атрибут и поддерживать его значения на уровне приложений. В большинстве случаев так и поступают. Так, в стандарте SQL появились специальные типы данных: *date* и *time*. Но в таком подходе имеются недостатки: СУБД не знает семантики временного поля отношения и не может контролировать корректность его значений. В связи с этим появляется дополнительная избыточность хранения (предыдущее состояние объекта хранится и в основной БД, и в журнале изменений).

Существует отдельное направление исследований и разработок в области так называемых темпоральных БД, где исследуются вопросы моделирования данных, языки запросов, организация данных во внешней памяти и т.д. Основные темпоральные системы на том принципе, что для любого объекта данных, созданного в момент времени t_1 и уничтоженного в момент времени t_2, в БД сохраняются (и доступны пользователям) все его состояния во временном интервале $[t_1, t_2]$.

Создание прототипов темпоральных СУБД обычно выполняется на основе некоторой реляционной СУБД в виде надстройки над реляционной системой.

Контрольные вопросы

1. Назовите основные направления совершенствования реляционных баз данных.
2. В чем заключается метод генерации систем баз данных?
3. Перечислите способы оптимизации запросов.
4. Для решения каких задач применяются темпоральные запросы?

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЕ СУБД

15.1. Общие понятия объектно-ориентированного подхода к разработке СУБД

Создание объектно-ориентированных баз данных началось в середине 1980-х гг. Наиболее активно ООБД развиваются в последние годы.

Развитие ООБД определяется, прежде всего, потребностями практики — необходимостью разработки сложных информационных прикладных систем.

Конечно, ООБД возникли не на пустом месте. Соответствующий базис обеспечили как предыдущие работы в области БД, так и развивающиеся языки программирования с абстрактными типами данных и объектно-ориентированные языки программирования.

Что касается связи с предыдущими работами в области БД, то, на наш взгляд, наиболее сильное влияние на развитие ООБД оказывают проработки реляционных СУБД и создаваемое на их основе семейство БД, поддерживающих управление сложными объектами. Кроме того, исключительное влияние на идеи и концепции создания ООБД оказало семантическое моделирование данных. Свое влияние оказывает также развитие параллельно с ООБД дедуктивных и активных БД.

В наиболее общей постановке объектно-ориентированный подход базируется на следующих концепциях:

- объект и идентификатор объекта;
- атрибут и метод;
- класс;
- иерархия и наследование классов.

Любая сущность реального мира в объектно-ориентированных языках и системах моделируется в виде объекта. Любой объект при своем создании получает генерируемый системой уникальный идентификатор, который связан с объектом все время его существования и не меняется при изменении состояния этого объекта.

Каждый объект характеризуют состояние и поведение. Состояние объекта — набор значений его атрибутов. Поведение объекта — набор методов (программный код), оперирующих его состоянием. Значение атрибута объекта — это тоже некоторый объект или множество объектов. Состояние и поведение инкапсулированы в объекте. Взаимодействуют объекты на основе передачи сообщений и выполнения соответствующих методов.

Множество объектов с одним и тем же набором атрибутов и методов образуют класс объектов. Объект должен принадлежать только одному классу (если не учитывать возможности наследования). Допускается наличие примитивных предопределенных классов, объекты-экземпляры которых не имеют атрибутов: целые, строки и т.д. Класс, объекты которого могут служить значениями атрибута объектов другого класса, называется *доменом* этого атрибута.

Допускается создание нового класса на основе уже существующего класса — наследование. В этом случае новый класс, называемый подклассом существующего класса (суперкласса), наследует все атрибуты и методы суперкласса. В подклассе, кроме того, могут быть определены дополнительные атрибуты и методы. Различают случаи простого и множественного наследования. В первом случае подкласс может определяться только на основе одного суперкласса, а во втором случае суперклассов может быть несколько. Если в языке или системе поддерживается единичное наследование классов, то набор классов образует древовидную иерархию. При поддержании множественного наследования классы связаны в ориентированный граф с корнем, называемый решеткой классов. Объект подкласса считается принадлежащим любому суперклассу этого класса.

Одной из более поздних идей объектно-ориентированного подхода к созданию СУБД является переопределение атрибутов и методов суперкласса в подклассе (перегрузка методов). Это увеличивает гибкость системы проектирования, но порождает дополнительную проблему: при компиляции объектно-ориентированной программы могут быть неизвестны структура и программный код описаний объекта, хотя его класс (в общем случае — суперкласс) известен. Для разрешения этой проблемы применяется так называемый метод позднего связывания, означающий интерпретационный режим выполнения программы с распознаванием деталей реализации объекта во время выполнения посылки сообщения к нему. Введение некоторых ограничений на способ определения подклассов позволяет добиться эффективной реализации СУБД.

При таком наборе базовых понятий, если не принимать во внимание возможности наследования классов и соответствующие проблемы, объектно-ориентированный подход очень близок к языкам программирования с абстрактными (произвольными) типами данных.

С другой стороны, если абстрагироваться от поведенческого аспекта объектов, объектно-ориентированный подход аналогичен семантическому моделированию данных. Фундаментальные абстракции, лежащие в основе семантических моделей, неявно используются и в объектно-ориентированном мо-

делировании. В состав сложных объектов могут входить и другие объекты со своими значениями атрибутов. Абстракция группирования — основа формирования классов объектов. На абстракциях специализации (обобщения) основано построение иерархии или решетки классов.

Наиболее важным новым качеством ООБД, которое позволяет достичь объектно-ориентированный подход, является поведенческий аспект объектов. В прикладных информационных системах, основывавшихся на БД с традиционной организацией (вплоть до тех, которые базировались на семантических моделях данных), существовал принципиальный разрыв между структурной и поведенческой частями. Структурная часть системы поддерживалась всем аппаратом БД, ее можно было моделировать, изменять, но поведенческая часть при этом создавалась изолированно. В частности, отсутствовали формальный аппарат и системная поддержка совместного моделирования и гарантирование согласованности структурной (статической) и поведенческой (динамической) частей. В среде ООБД проектирование, разработка и сопровождение прикладной системы становятся процессом, в котором интегрируются структурный и поведенческий аспекты. Конечно, для этого требуются специальные языки, позволяющие определять объекты и создавать на их основе прикладную систему.

Специфика применения объектно-ориентированного подхода к организации и управлению БД потребовала уточнения толкования классических концепций и некоторого их расширения. Это определяется потребностью долговременного хранения объектов во внешней памяти, ассоциативным доступом к объектам, обеспечением согласованного состояния ООБД в условиях мультидоступа и тому подобными возможностями, свойственными базам данных. Выделяют три аспекта, отсутствующих в традиционном методе проектирования БД, но требующихся в ООБД.

Первый аспект касается потребности в средствах спецификации знаний при определении класса (ограничений целостности, правил дедукции и т. п.).

Второй аспект — потребность в механизме определения разного рода семантических связей между объектами разных классов. Фактически это означает требование полного распространения на ООБД средств семантического моделирования данных. Потребность в использовании абстракции ассоциирования отмечается и в связи с использованием ООБД в сфере автоматизированного проектирования.

Третий аспект связан с пересмотром понятия класса. В контексте ООБД оказывается более удобным рассматривать класс как множество объектов данного типа, т. е. одновременно поддерживать понятия и типа, и класса объектов.

15.2. Объектно-ориентированные модели данных

Первой формализованной и общепризнанной моделью данных была реляционная модель Кодда. В этой модели, как и во всех следующих, выделялись три аспекта: структурный, целостный и манипуляционный. Структуры данных в реляционной модели основываются на плоских нормализованных отношениях, ограничения целостности выражаются с помощью средств логики первого порядка, манипулирование данными осуществляется на основе реляционной алгебры или равносильного ей реляционного исчисления. Своим успехом реляционная модель данных во многом обязана тому, что она опирается на строгий математический аппарат реляционной алгебры и теории множеств.

Основные трудности объектно-ориентированного моделирования данных связаны с тем, что не существует конкретного математического аппарата, на который могла бы опираться общая объектно-ориентированная модель данных. Разработка методов управления данными внутри объектов, как и любой процесс программирования нетрадиционных задач, остается «искусством программирования».

Методы могут быть *публичными* (доступными из объектов других классов) или *приватными* (доступными только внутри данного класса).

Итак, объектно-ориентированная система управления базами данных представляет собой объединение системы программирования и СУБД и основана на объектно-ориентированной модели данных.

Основное назначение ООБД связано с потребностью создания единого информационного пространства.

В этой среде должны отсутствовать противоречия между структурной и поведенческой частями проекта и должно поддерживаться эффективное управление сложными структурами данных во внешней памяти.

В отличие от традиционных реляционных систем, в которых при создании приложения приходится одновременно использовать процедурный язык программирования, ориентированный на работу со скалярными значениями, и декларативный язык запросов, ориентированный на работу с множествами, языковая среда ООБД — это объектно-ориентированная система программирования, естественно включающая в себя средства работы с долговременными объектами. Естественность включения средств работы с БД в язык программирования означает, что работа с долговременными (хранимыми во внешней БД) объектами должна происходить на основе тех же синтаксических конструкций (и с той же семантикой), что и работа с временными объектами, существующими только во время работы программы.

Эта сторона ООБД наиболее близка родственному направлению языков программирования баз данных. Языки программирования ООБД и БД во многом различаются только терминологически; существенным отличием является лишь поддержание в языках ООБД подхода к наследованию классов.

Другим аспектом языкового окружения ООБД является потребность в языках запросов, которые можно было бы использовать в интерактивном режиме. Если доступ к объектам внешней БД в языках программирования ООБД носит в основном навигационный характер, то для языков запросов более удобен декларативный стиль. Как известно, декларативные языки запросов менее развиты, чем языки программирования.

15.3. Языки программирования объектно-ориентированных баз данных

К настоящему моменту неизвестен какой-либо язык программирования ООБД, который был бы спроектирован целиком заново, начиная с нуля. Естественным подходом к построению языков программирования ООБД было использование (с необходимыми расширениями) некоторого существующего объектно-ориентированного языка.

Одним из первых созданных для реализации объектно-ориентированного и функционального подходов к программированию является язык Лисп (Common Lisp).

Потребность более эффективной реализации заставляет в качестве основы объектно-ориентированного языка использовать известные языки программирования Basic и Си++.

Потребность в поддержании в объектно-ориентированной СУБД не только языка (или семейства языков) программирования ООБД, но и развитого языка запросов в настоящее время признается практически всеми разработчиками. Система должна поддерживать легко осваиваемый интерфейс, прямо доступный конечному пользователю в интерактивном режиме.

Наиболее распространенный подход к организации интерактивных интерфейсов с объектно-ориентированными системами баз данных основывается на использовании так называемых обходчиков. В этом случае конечный интерфейс обычно является графическим. На экране отображается схема (или подсхема) ООБД, и пользователь осуществляет доступ к объектам в навигационном стиле. Некоторые исследователи считают, что в этом случае разумно игнорировать принцип инкапсуляции объектов и предъявлять пользователю свойства объектов. В большинстве существующих систем ООБД подобный интерфейс существует, но всем понятно, что навигационный язык запросов — это в некотором смысле

шаг назад по сравнению с языками запросов даже реляционных систем.

Ненавигационные языки запросов. В настоящее время существует три подхода к разработке таких языков.

Первый подход заключается в расширении языков запросов реляционных систем. Наиболее распространены языки с синтаксисом, близким к языку SQL. Это связано, конечно, с общим признанием и чрезвычайно широким распространением этого языка.

Второй подход основывается на построении полного логического объектно-ориентированного языка исчисления.

Третий подход основывается на применении декларативного и объектно-ориентированного методов программирования.

Независимо от применяемого при создании языка запросов подхода перед разработчиками встает одна концептуальная проблема, решение которой не укладывается в традиционный объектно-ориентированный подход.

Исходя из концепции ООБД основой для формулирования запроса должен служить класс, представляющий в ООБД множество однотипных объектов. Но что может представлять собой результат запроса? Набор основных понятий объектно-ориентированного подхода не содержит подходящего к данному случаю ответа. Обычно из положения выходят, расширяя базовый набор концепций множества объектов и полагая, что результатом запроса является некоторое подмножество объектов — экземпляров класса. Это довольно ограниченный подход, поскольку автоматически исключается возможность наличия в языке запросов средств, аналогичных реляционному оператору соединения.

Проблемы оптимизации запросов. Основной целью оптимизации запроса в системе ООБД является создание оптимального плана его выполнения с использованием доступа к внешней памяти данной БД.

Оптимизация запросов хорошо исследована и разработана в контексте реляционных БД. Известны методы синтаксической и семантической оптимизации на уровне непроцедурного представления запроса, алгоритмы выполнения элементарных реляционных операций, методы оценок стоимости планов запросов.

Конечно, объекты могут иметь существенно более сложную структуру, чем кортежи плоских отношений, но не это различие является наиболее важным. Основная сложность оптимизации запросов к ООБД следует из того, что условия выборки формулируются в терминах внешних атрибутов объектов (методов), а для реальной оптимизации (т.е. для выработки оптимального плана) требуются условия, определенные внутренними атрибутами (переменными состояния).

Похожая ситуация существует и в реляционных СУБД при оптимизации запроса к БД. В этом случае условия также формулиру-

ются в терминах внешних атрибутов (атрибутов представления), и в целях оптимизации запроса эти условия должны быть преобразованы в условия, определенные атрибутами хранимых отношений. Хорошо известным методом такой предоптимизации является подстановка представлений, что часто (хотя и не всегда в случае использования языка SQL) обеспечивает требуемые преобразования.

В системах ООБД ситуация существенно усложняется двумя обстоятельствами.

Во-первых, методы предоптимизации обычно программируются на некотором процедурном языке и могут иметь параметры, т.е. в общем случае тело метода представляет собой не просто арифметическое выражение, как в случае определения атрибутов представления, а параметризованную программу, содержащую ветвления, вызовы функций и методы описания других объектов.

Во-вторых, точная реализация метода и даже структура объекта могут быть неизвестны во время компиляции запроса.

Одним из подходов к упрощению проблемы оптимизации является открытие видимости некоторых (наиболее важных для этого) внутренних атрибутов объектов. В этом случае достаточно открыть некоторые внутренние атрибуты только для компилятора запросов, т.е. фактически запретить переопределять эти переменные в подклассах. С точки зрения пользователя открытые атрибуты имеют вид переменных без параметров, возвращающих значение соответствующего типа. Однако лучше было бы сохранить строгую вложенность объектов (чтобы избавить приложение от критической зависимости от реализации) и обеспечить возможность тщательного проектирования схемы ООБД с учетом потребностей оптимизации запросов.

Общий подход к предоптимизации условий выборки для одного класса объектов может быть следующим.

1. Преобразовать логическую формулу условия выборки в конъюнктивную нормальную форму (КНФ). Не будем останавливаться на способе выбора конкретной КНФ, но, естественно, она должна быть хорошей (например, содержащей максимальное число атомарных конъюнктов).

2. Для каждого конъюнкта, включающего в себя методы с известным во время компиляции телом, заменить вызовы методов на их тела с подставленными параметрами. (Будем предполагать, что параметры не содержат вызовов функций или методов других объектов.)

3. Для каждого конъюнкта произвести все возможные упрощения, т.е. вычислить все, что можно вычислить в статике.

4. Если получены конъюнкты, представляющие собой простые предикаты сравнения на основе переменных состояния и констант, следует использовать их для выработки оптимального пла-

на выполнения запроса. Если же такие конъюнкты получить не удалось, единственным способом удаления класса объектов является его последовательный просмотр с полным вычислением логического выражения для каждого объекта.

Указанные ограничения не вызывают зависимости прикладной программы от особенностей реализации ООБД. Это обусловлено тем, что объекты БД могут быть определены семантически.

Контрольные вопросы

1. Назовите принципы объектно-ориентированного подхода к созданию баз данных.
2. Какие объектно-ориентированные модели данных вы знаете?
3. Какие языки программирования применяют для разработки объектно-ориентированных баз данных?

ГЛАВА 16

ОБЪЕКТНО-ОРИЕНТИРОВАННАЯ СУБД CACHE

16.1. Структура СУБД Cache

СУБД Cache относится к постреляционным объектно-ориентированным системам. Термин *постреляционная СУБД* означает принадлежность к системам нового поколения. Имеется в виду не столько аспект времени (СУБД появилась после своих основных реляционных конкурентов), сколько ряд технологических новшеств, таких как единая архитектура данных и полная поддержка объектно-ориентированных технологий.

В соответствии с принципами проектирования объектно-ориентированных баз данных система Cache:

- содержит объект — элемент БД, в котором хранятся не только данные, но и методы их обработки;
- позволяет обрабатывать мультимедийные данные и предоставляет пользователям возможность создавать собственные структуры данных любой сложности;
- допускают работу на высоком уровне абстракции.

Отличительной особенностью СУБД является независимость хранения данных от способа их представления, что реализуется с помощью так называемой единой архитектуры данных. В рамках данной архитектуры существует единое описание объектов и таблиц, отображаемых непосредственно в многомерные структуры ядра базы данных, ориентированных на обработку транзакций. Как только определяется класс объектов, автоматически генерируется реляционное описание данных этого класса в формате SQL; как только в *Словарь данных* поступает язык определения данных (ЯОД) — описание в формате реляционной базы данных, автоматически генерируется реляционное и объектное описание данных, т.е. устанавливается доступ к ним в формате объектов. При этом все описания ведутся согласованно и все операции по редактированию проводятся только с одним описанием данных. Это позволяет сократить время разработки и сэкономить вычислительные ресурсы. Приложения будут работать значительно быстрее.

Cache — многоплатформенная система, которая поддерживает следующие операционные системы: всю гамму ОС Windows, Linux, основные реализации Unix и Open VMS. Планируется также поддержка новых реализаций Unix. Большое внимание уделяется новой платформе Itanium.

Данные в Cache хранятся под управлением многомерного сервера данных (MDS). В основе Cache лежит транзакционная многомерная модель данных, которая позволяет хранить и представлять данные в таком виде, в котором они чаще всего используются. Многомерный сервер данных снимает многие ограничения, накладываемые реляционными СУБД, хранящими данные в двухмерных таблицах. Как известно, реляционная модель БД состоит из большого числа таблиц, что необходимо при работе со сложными структурами данных. Это, в свою очередь, существенно усложняет и замедляет выполнение сложных транзакций и ведет к хранению излишней информации. Система Cache хранит данные в виде многомерных разреженных массивов.

Уникальная транзакционная многомерная модель данных позволяет избежать проблем, присущих реляционным СУБД, оптимизируя данные уже на уровне хранения.

Многомерный сервер данных Cache предназначен для обработки транзакций в системах с большими и сверхбольшими базами данных (сотни гигабайт, терабайты) и большим числом одновременно работающих пользователей. Многомерный сервер данных Cache обеспечивает высокую производительность системы за счет отказа от хранения избыточных данных и таблиц.

Транзакционная модель данных Cache позволяет оптимизировать данные на уровне хранения, поддерживать объектную модель и сложные типы данных. Все эти возможности значительно упрощают создание сложных систем.

В Cache реализована концепция единой архитектуры данных, т.е. к одним и тем же данным, хранящимся под управлением многомерного сервера данных Cache, есть три способа доступа: прямой, реляционный и объектный.

Прямой доступ к данным (*Cache Direct Access*) обеспечивает максимальную производительность СУБД и полный контроль работы системы со стороны программиста. Разработчики приложений получают возможность работать напрямую со структурами хранения. Использование этого типа доступа накладывает определенные требования на квалификацию разработчиков, позволяет оптимизировать хранение данных приложения и создавать сверхбыстрые алгоритмы обработки данных.

Реляционный доступ к данным (*Cache SQL*) обеспечивает максимальную производительность реляционных приложений с использованием встроенного языка SQL. Cache SQL соответствует стандарту SQL. Кроме того, разработчик может использовать разные типы триггеров и хранимых процедур.

Даже без использования прямого и объектного доступа к данным приложения в Cache работают быстрее за счет высокой производительности многомерного сервера данных.

Объектный доступ к данным (*Cache Objects*) осуществляется при использовании объектно-ориентированных языков программирования Java, Visual C++, VB и других ActiveX-совместимых средств разработки, таких как PowerBuilder и Delphi. Для этого в Cache реализована объектная модель управления базами данных, в которой полностью поддерживаются наследование признаков (в том числе и множественное), инкапсуляция и полиморфизм. При создании информационной системы разработчик получает возможность использовать объектно-ориентированный подход к разработке, моделируя предметную область в виде совокупности классов объектов, в которых хранятся данные (свойства классов) и поведение классов (методы классов).

Система Cache, поддерживая объектную модель данных, позволяет естественным образом использовать объектно-ориентированный подход как при проектировании предметной области, так и при реализации приложений средствами разработки (Java, C++, Delphi, VB).

Как только определяется класс объектов, Cache автоматически генерирует реляционное описание этих объектов таким образом, чтобы к ним можно было обращаться, используя SQL.

Аналогично при импорте в *Словарь данных* (описаний реляционной базы данных) Cache автоматически генерирует реляционное и объектное описания данных, открывая тем самым доступ к ним, как к объектам. При этом все описания данных ведутся согласованно, а все операции по редактированию проводятся только с одним экземпляром данных. Кроме того, возможен прямой доступ программиста к этим данным.

Система Cache позволяет комбинировать три типа доступа к данным, т.е. оставляет разработчику свободу выбора. Например, при реализации системы объектный доступ может использоваться при описании бизнес-логики приложения и создании пользовательского интерфейса с помощью объектно-ориентированных средств разработки (VB, Delphi, C++).

Реляционный доступ может применяться для обеспечения совместимости с другими системами, интеграции с инструментами построения отчетов и аналитической обработки данных (Seagate Info, Cognos, Business Objects).

Прямой доступ к данным может быть использован при реализации таких операций, в которых применение обычных хранимых процедур, основанных на SQL, не может обеспечить необходимую производительность. Использование прямого доступа для реализации сложных операций позволяет увеличить производительность системы на один-два порядка.

Для реализации бизнес-логики в СУБД используется специальный язык Cache Object Script (COS) — полнофункциональный язык, который имеет все необходимые механизмы для работы с данны-

ми при любом способе доступа. С помощью COS разработчик создает методы классов, триггеры, хранимые процедуры, различные служебные программы. Стоит отметить интерфейсы со средствами проектирования и разработки приложений. Специальные компоненты Cache позволяют проектировать приложения как для объектного, так и для реляционного методов обработки данных.

Кроме того, поддерживаются следующие интерфейсы: Native C++, Java, EJB, ActiveX, XML, а также интерфейсы CallIn и CallOut.

Для обеспечения надежности в Cache предусмотрены такие механизмы, как журналы до и после записи, теневой сервер, репликация, «горячее» резервное копирование и т.д.

Протокол распределенного кэша (Cache Distibuted Cache Protocol) — уникальная сетевая технология фирмы InterSystems, которая распределяет базу данных по сети, оптимизируя ее производительность и пропускную способность в зависимости от работы приложений.

Cache — открытая система, в которой поддерживается множество интерфейсов к средствам проектирования и разработки приложений.

Cache работает практически на всех популярных платформах с наиболее распространенными Web-серверами. При этом обеспечивается полная переносимость приложений с платформы на платформу.

При всех достоинствах современной объектно-ориентированной технологии разработки баз данных имеется несколько препятствий, которые удерживают разработчиков от принятия решения о переходе на нее с реляционной технологии. Основным препятствием является значительный объем имеющихся разработок, опирающихся на реляционные СУБД. При переходе на новую технологию обработки данных необходимо многое начинать с нуля, поэтому возникает вопрос целесообразности такого перехода.

Кроме того, объектная технология, поддерживаемая в ряде постреляционных СУБД, не имеет развитого и стандартизированного языка генерации отчетов и анализа данных, каким является структурированный язык запросов SQL.

В системе Cache данные проблемы были решены за счет процедур, обеспечивающих возможность перехода с реляционной технологии на объектную.

16.2. СУБД Cache и Web-технологии

Одним из условий разработки и внедрения на предприятиях CALS-технологий являются Web-сервисы, которые и предоставляет среда Cache.

Как известно, основной принцип работы Web-среды состоит в том, что все документы в ней создаются в едином формате HTML, а клиентские приложения — броузеры, функционирующие на самых различных компьютерных платформах, почти одинаково отображают эти документы. Однако решения отражают статические принципы обработки документов. Для придания им динамики требуется использование различных средств программирования, действующих как на стороне клиента, так и на стороне сервера.

Если разработчик применял скриптовые языки, функционирующие на стороне клиента, то область их действия была ограничена отображаемым документом, а в качестве входных данных использовались либо действия пользователя, либо информация, хранящаяся в его компьютере. Это ограничение связывалось с тем, что скриптовые языки не получали какой-либо дополнительной информации от Web-сервера, с которого был загружен документ, вместе с которым и были получены скрипты.

Если на сайте требовалось использовать несколько более сложные способы управления данными, то приходилось применять исполняемые модули, функционирующие на самом сервере. Только таким образом можно было осуществлять доступ к базам данных и на основе полученной информации формировать страницы, передаваемые затем удаленному пользователю. Только на сервере удавалось обеспечить поддержку сеансов работы и идентификацию пользователей, столь необходимые во всех приложениях электронной коммерции.

Для реализации Web-технологий использовались и классические языки, такие как C++, и языки, специально ориентированные на Web-среду, такие как ASP. Однако все они имели одну общую черту — конечный вывод документов производился в формате HTML.

Если учитывать общую скорость прогресса в компьютерной индустрии, то можно признать, что долгожитель HTML, который достаточно долго использовался в сети Интернет, может в ближайшем будущем быть заменен новым языком XML. Более того, международные стандарты по CALS-технологиям уже ориентируются на этот язык форматирования документов.

Система Web-сервисов и обособленных клиентов нужна только для ресурсов с чрезвычайно сложной функциональностью.

В общем случае Web-сервисы принимают и передают информацию на языке XML. Этот язык, естественно, является открытым стандартом. Также следует учитывать, что XML, как и его предшественник HTML, не зависит от платформы и операционной системы. Сочетание этих двух факторов позволяет разработчикам свободно создавать самые различные клиентские приложения, функционирующие на разных компьютерных платформах,

т.е. одному сервису может соответствовать несколько клиентских приложений.

XML-документы, пересылающиеся от сервера к клиенту и обратно, передаются по протоколу HTTP, который пропускается практически всеми брандмауэрами, что также прибавляет привлекательности идее Web-сервисов.

Следует отметить и тот факт, что Web-сервисы являются самодокументируемыми, т.е. любое клиентское приложение может получить информацию о структуре сервиса, его функциональности и правилах вызова функций, поддерживаемых Web-сервисом.

Web-сервисы способны передавать и получать информацию тремя способами: с применением методов GET и POST стандартного протокола HTTP или с помощью языка SOAP, который является производным от XML. Первые два варианта разрешают использовать в качестве клиентского приложения стандартный броузер, но необходимо признать, что это далеко не идеальный вариант. Во-первых, с помощью броузера трудно организовать вызов всех функций достаточно сложного сервиса. Разработчику необходимо исследовать структуру сервиса заранее, т.е. перед тем как создавать Web-страницы для доступа к нему, и в этом случае теряются преимущества самодокументируемости. Во-вторых, следует помнить, что вывод информации все равно будет идти в «чистом» XML, который броузер не сможет адекватно отобразить. Поэтому для работы с Web-сервисами, обладающими достаточно серьезной функциональностью, удобно использовать язык SOAP.

16.3. Среда разработки приложений Visual Basic.NET

Visual Basic.NET — современная визуальная среда быстрой разработки Windows-приложений, создания полнофункциональных профессиональных проектов на базе современной объектно-ориентированой концепции. Конструирование пользовательского интерфейса в Visual Basic.NET сведено к работе с мышью. Иногда требуется лишь добавить несколько строк кода, созданного мастером проекта. Visual Basic.NET — это достаточно простой, строго типизированный язык, с помощью которого легко строить объектно-ориентированные конструкции, проектировать графический пользовательский интерфейс (GUI), работать с базами данных.

Таким образом, Visual Basic.NET, интегрированный с Cache, обеспечивает:

- простоту создания пользовательского интерфейса программы;
- возможность работы с Web-сервисами;
- создание клиент-серверных приложений, включая работу через Интернет;
- поддержку SOAP-протокола.

16.4. Многоплатформенный протокол передачи данных SOAP

SOAP-протокол — это набор правил для приложений, которые могут вызывать работы с удаленными объектами. Где именно находятся эти удаленные объекты — в другом каталоге, где-то в корпоративной интрасети или в Интернете — для клиентских программ, использующих SOAP, абсолютно неважно. SOAP основан на языке XML. Каждая передача информации между клиентом и сервером является отдельным XML-документом, который написан по правилам SOAP.

Как известно, логическая структура каждого XML-документа определяется его DTD-блоком. Для SOAP заранее определены все возможные теги и типы данных, поэтому SOAP-документы не нуждаются в DTD-блоках. За счет подобной унификации серверы и клиенты освобождаются от достаточно тяжелой процедуры синтаксического анализа приходящего документа с не определенным заранее набором тегов.

SOAP-протокол — это слабосвязанный механизм, ориентированный на сообщения и предназначенный для удаленного вызова объектов по глобальным сетям.

Рассмотрим принцип его работы. Зная URL, клиент на языке описания сервиса SDL (Service Description Language) запрашивает у сервера информацию о нем (предоставляемые методы, параметры и т.д.). В ответ клиент получает SDL-файл с необходимыми сведениями о том, какие услуги (методы) ему будут предоставляться и как ими пользоваться.

Обращения к объектам БД осуществляются с помощью HTTP-запросов и ответов. В запрос вкладывается XML-текст, состоящий из трех частей:

- пакет (envelope) SOAP, определяющий содержимое сообщения;
- заголовок SOAP, определяющий возможные заголовки сообщения;
- тело SOAP, содержащее информацию о запросе или ответе на запрос.

Ответ приходит также в виде XML, содержащего результаты обработки запроса или код ошибки.

Новая версия Cache — Cache 5 — реализует SOAP-протокол для всех поддерживаемых платформ. При этом не требуется использование промежуточных приложений сторонних компаний, все необходимые функции реализованы в виде системных классов БД Cache.

В Cache SOAP-протокол предоставляет широкий спектр функций, который включает в себя:

- создание Web-служб путем определения классов, содержащих Web-методы, работы с удаленными системами. При этом ме-

тоды вызываются непосредственно в среде БД, что приводит к увеличению производительности системы за счет сокращения трафика между клиентом и сервером;

- автоматическое создание и публикацию каталога доступных методов (WSDL).

Для создания Web-службы и публикации каталога на сервере не требуется никаких дополнительных знаний о системе Cache. Реализация служб осуществляется на основе объектной модели Cache. При этом для создания службы необходимо определить класс, наследуемый от системного класса %SOAP.WebService. Методы, которые подлежат публикации на сервере, характеризуются параметром WebMethod. Код, реализующий логику метода, как и код обыкновенного метода класса Cache, может быть реализован с использованием Cache Object Script, Cache Basic или Cache SQL. Возможно использование сложных типов данных (встраиваемых объектов, коллекций, отношений и др.) в качестве аргументов и возвращаемого значения публикуемого метода. При этом Cache автоматически создает необходимые объекты сложных типов данных при поступлении соответствующего запроса на сервер.

Приведем пример Web-службы с одним объектом:

```
Class MyApp.StockService Extends %SOAP.WebService
{
Parameter SERVICENAME = 'MyStockService';
Parameter LOCATION = 'http://localhost/csp/user';
Parameter NAMESPACE = 'http://tempuri.org';
/// Метод возвращает цену на завтра
ClassMethod Forecast (StockName As %String) As
%Integer [WebMethod]
{
// применяем генератор случайных чисел
Set price = $Random(1000)
Quit price
}
}
```

Кроме того, Cache автоматически генерирует тестовые CSP-страницы методов и CSP-страницу каталога, которые можно использовать для проверки правильной работы создаваемой службы. При этом CSP-страницы представляются в виде обыкновенных Web-приложений, доступных для просмотра обычным броузером.

В Cache SOAP-сервер работает следующим образом:

- для каждого Web-service создается новый Cache-класс, который расширяется с помощью %SOAP.WebService;
- с помощью нового Cache-класса определяется один или более методов, соответствующих методам Web-service, каждый из которых может быть определен ключевым словом WebMethod в

его описании. Можно также определить Web-методы, которые возвращают массивы объектов, с помощью объявления запроса и добавления ключевого слова WebMethod в его описании;

• компилируется класс Web-service , при этом компилятор Cache автоматически привязывает каталог с информацией, описывающей содержимое SOAP-service, и строит SOAP-интерфейс для каждого Web-метода. SOAP-интерфейс для Web-метода является сгенерированным классом, который выполняет преобразование запроса SOAP в специфический вызов Web-метода, используя XML-технологию Cache;

• Cache автоматически создает WSDL-документ для каждого Web-service. WSDL-документ является XML-документом. WSDL-документ создается с помощью Web (HTTP)-сервера, использующего CSP. WSDL-документ является динамически создаваемым и будет автоматически отображать любые изменения класса Web-service;

• SOAP-client открывает доступный Web-service, запрашивая WSDL-документ, который, в свою очередь, посылает запрос Cache-серверу. Используя эту информацию в WSDL-документе, SOAP-client активизирует определенный метод с помощью создания XML-сообщения и посылки его (с помощью HTTP) SOAP-серверу;

• SOAP-сервер Cache получает вызов SOAP с помощью Cache (CSP) HTTP Gateway. Сервер распаковывает сообщение, проверяет его правильность и вызывает определенный Web-метод. Перед вызовом Web-метода SOAP-сервер Cache конвертирует все входные параметры к соответствующему представлению Cache;

• Web-метод выполняет свой код и возвращает ответ. Этот ответ может быть простой строковой константой, может быть XML-объектом или массивом.

Система Cache обеспечивает возможность создания класса SOAP-service client, содержащего методы, которые вызывает Web-service, используя SOAP-протокол.

Для каждого Web-service (набора связанных методов SOAP), который вы желаете вызвать, необходимо создать новое определение класса Cache, которое получено от %SOAP.WebClient, находящегося в библиотеке Cache.

Класс SOAP-client содержит один или более методов, соответствующих методам Web-service.

При компиляции класса SOAP-client транслятор класса Cache автоматически транслирует информацию каталога, описание содержания SOAP и создает интерфейс SOAP-client для каждого Web-service;

Интерфейс SOAP-client для Web-service — это сгенерированный класс, который исполняет работу преобразования запроса SOAP в определенный запрос Web-service, используя технологию перевода объекта в формат XML.

SOAP-service обнаруживает доступный Web-service с помощью запроса WSDL-документа от Web-сервера, который, в свою очередь, запрашивает его у Cache-сервера. Используя эту информацию в WSDL-документе, SOAP-client вызывает определенный метод, создавая XML-сообщение и отправляя его через HTTP на SOAP-сервер, как определено в WSDL-документе.

SOAP-сервер получает запрос SOAP, распаковывает сообщение, проверяет его и вызывает указанную операцию.

Контрольные вопросы

1. Чем принципиально отличается СУБД Cache от реляционных СУБД?
2. Как осуществляется в системе Cache прямой доступ к данным?
3. Какие протоколы передачи информации в сети Интернет поддерживаются в системе Cache?
4. Перечислите особенности среды разработки приложений Visual Basic.NET.

ГЛАВА 17

СИСТЕМЫ БАЗ ДАННЫХ, ОСНОВАННЫЕ НА ПРАВИЛАХ

17.1. Структура базы данных

В базе данных хранится три вида информации.

1. Информация, характеризующая структуры пользовательских данных (описание структурной части схемы базы данных). В реляционных базах данных такая информация сохраняется в системных отношениях-каталогах и содержит главным образом имена базовых отношений, имена и типы данных их атрибутов.

2. Наборы кортежей пользовательских данных, сохраняемых в определенных пользователями отношениях.

3. Правила, определяющие ограничения целостности базы данных, триггеры базы данных и представляемые (виртуальные) отношения.

В реляционных системах правила также сохраняются в системных таблицах-каталогах, хотя плоские таблицы далеко не идеально подходят для этой цели.

Информация первого и второго видов в совокупности явно описывает объекты (сущности) реального мира, моделируемые в базе данных. Другими словами, это явные факты, предоставленные пользователями для хранения в БД. Эту часть базы данных принято называть *экстенциональной*.

Информация третьего вида служит для руководства СУБД при выполнении различного рода операций, задаваемых пользователями.

Ограничения целостности могут блокировать выполнение операций обновления базы данных, триггеры вызывают автоматическое выполнение специфицированных действий при возникновении специфицированных условий, определения представлений вызывают явную или косвенную материализацию представляемых таблиц при их использовании. Эту часть базы данных принято называть *интенциональной*; она содержит не непосредственные факты, а информацию, характеризующую семантику предметной области.

В реляционных базах данных наиболее важное значение имеет экстенциональная часть, а интенциональная часть играет в основном вспомогательную роль.

В системах баз данных, основанных на правилах, эти две части, как минимум, равноправны.

17.2. Активные базы данных

База данных называется активной, если СУБД по отношению к ней выполняет не только те действия, которые явно указывает пользователь, но и дополнительные действия в соответствии с правилами, заложенными в саму БД.

Основа идеи создания активной БД содержалась в языке SQL. Действительно, есть определение триггера или условного воздействия, как не введение в БД правила, в соответствии с которым СУБД должна производить дополнительные действия. Плохо лишь то, что триггеры не были полностью реализованы ни в одной из известных систем. И это не случайно, потому что реализация такого аппарата в СУБД очень сложна и не совсем понятна.

Среди вопросов, ответы на которые до сих пор не получены, следующие.

Как эффективно определить набор вспомогательных действий, вызываемых прямым действием пользователя?

Каким образом распознавать циклы в цепочке *действие—условие—действие—...* и что делать при возникновении таких циклов?

В рамках какой транзакции выполнять дополнительные условные действия и к бюджету какого пользователя относить возникающие накладные расходы?

Масса проблем не решена даже для сравнительно простого случая реализации триггеров SQL, а задача ставится уже гораздо шире. Предлагается иметь в составе СУБД продукционную систему общего вида, условия и действия которой не ограничиваются содержимым БД или прямыми действиями с ее данными пользователя. Условие может содержать время суток, а действие может быть внешним, например вывод информации на экран оператора. Практически все современные работы по активным БД связаны с проблемой эффективной реализации такой продукционной системы.

Намного важнее в практических целях реализовать в реляционных СУБД аппарат триггеров. В проекте стандарта SQL3 предусматривается существование языковых средств определения условных воздействий. Их реализация и будет первым практическим шагом к содержанию активных БД (уже имеются соответствующие коммерческие реализации).

17.3. Дедуктивные базы данных

Дедуктивная БД состоит из двух частей: экстенциональной, содержащей факты, и интенциональной, содержащей правила для логического вывода новых фактов на основе экстенциональной части и запроса пользователя.

При таком общем определении SQL-ориентированную реляционную СУБД можно отнести к дедуктивным системам. Действительно, что есть определенные в схеме реляционной БД представления, как не интенциональная часть БД. Не так уж важно, какой конкретный механизм используется для вывода новых фактов на основе существующих. В случае использования SQL основным элементом определения представления является оператор выборки языка SQL, что вполне естественно, поскольку с его помощью создается таблица. Обеспечивается при этом и необходимая расширяемость, поскольку представления могут определяться не только над базовыми таблицами, но и над запросами.

Основным отличием реальной дедуктивной СУБД от реляционной является то, что в ней и правила интенциональной части БД, и запросы пользователей могут содержать рекурсию. При этом можно спорить о том, что всегда ли это хорошо. С одной стороны, возможность определения рекурсивных правил и запросов обеспечивает простое решение в дедуктивных базах данных проблем, которые вызывают большие трудности в реляционных системах (например, проблемы разборки сложной детали на примитивные составляющие). С другой стороны, именно возможность рекурсии делает реализацию дедуктивной СУБД очень сложной и во многих случаях неразрешимой проблемой.

Не будем подробно рассматривать конкретные проблемы, применяемые ограничения и используемые методы в дедуктивных системах. Отметим лишь, что обычно языки запросов и определения интенциональной части БД являются логическими (поэтому дедуктивные БД часто называют логическими). Имеется прямая связь дедуктивных БД с базами знаний (интенциональную часть БД можно рассматривать как базу знаний — БЗ). Более того, трудно провести грань между этими двумя сущностями; по крайней мере, общего мнения по этому поводу не существует.

Какова же связь дедуктивных БД с реляционными СУБД, кроме того, что реляционная БД является частным случаем дедуктивной? Связь заключается в том, что для реализации дедуктивной СУБД обычно применяется реляционная система, которая выполняет роль хранителя фактов и исполнителя запросов, поступающих с уровня дедуктивной СУБД. Такое использование реляционных СУБД резко актуализирует задачу глобальной оптимизации запросов.

При применении реляционной СУБД запросы обычно поступают на обработку по одному, поэтому нет повода для их глобальной (межзапросной) оптимизации. Дедуктивная же СУБД при выполнении одного запроса пользователя в общем случае генерирует пакет запросов к реляционной СУБД, которые могут оптимизироваться совместно.

Конечно, когда набор правил дедуктивной БД становится большим и их невозможно разместить в оперативной памяти, возникает проблема управления их хранением и доступом к ним во внешней памяти. В этом случае также можно применить реляционную систему, но уже не очень эффективно, так как требуются более сложные структуры данных и другие условия выборки. Известны лишь частные попытки решить эту проблему, но общего решения пока нет.

Контрольные вопросы

1. Чем отличаются структуры таблиц баз данных, основанных на правилах, от традиционных (реляционных) БД?
2. Назовите основные характеристики активных и дедуктивных баз данных.

ГЛАВА 18

МНОГОПОЛЬЗОВАТЕЛЬСКИЕ СИСТЕМЫ УПРАВЛЕНИЯ ЖИЗНЕННЫМ ЦИКЛОМ ПРОДУКЦИИ

18.1. Интегрированная информационная среда предприятия

В подразд. 2.1 уже рассматривались новое направление развития информационных систем в условиях современного производства и бизнеса — CALS-технологии — и необходимость в связи с этим перехода к организации на предприятиях распределенных многопользовательских БД. Рассмотрим основные направления организации работ по созданию на любом предприятии единого информационного пространства.

Как следует из концептуальной модели этого направления работ, основой CALS-технологий и создаваемых на этой основе автоматизированных систем является интегрированная информационная среда.

Представление об ИИС было введено в научный обиход задолго до появления CALS-технологий. Еще в 1983 г. японский ученый Н. Окино опубликовал работу, в которой утверждал, что производство материальных объектов и сопутствующие ему процессы проектирования, технологической подготовки и управления так сильно отличаются от других видов деятельности человека, что им должна отвечать особая архитектура программно-методического, математического и информационного обеспечения.

По его мнению, принципиальная разница между обработкой информации в производственной системе и в других случаях применения вычислительной техники в основном сводится к двум положениям.

1. Производство и все процессы в нем принадлежат физическому миру, а процессы, протекающие в компьютере, — миру информации. Следовательно, необходимо преобразование производственных проблем в информационные, а также обратный переход из информационного мира в физический. Это проблема адекватного моделирования, т.е. установления соответствия (по возможности взаимно-однозначного) между физическим и информационным пространством. При создании традиционного математического обеспечения (МО) для решения вычислительных задач рассматривается единственная математическая модель этой проблемы, которая через прикладной интерфейс адаптируется к различным областям применения (рис. 18.1).

Такой подход к решению производственных проблем практически не реализуем, поскольку ввиду их сложности и многообразия единую модель создать невозможно.

При этом если добавить к изучавшимся Н.Окино производственным проблемам рассмотрение поставок, эксплуатации, обслуживания и ремонта изделий, т.е. все постпроизводственные стадии жизненного цикла изделия, то ситуация становится еще более сложной.

2. В связи с отмеченными ранее недостатками традиционного подхода к созданию МО (см. рис. 18.1) предлагается отбросить стратегию единственной модели организации баз данных и перейти к стратегии, сущность которой показана на рис. 18.2.

Здесь роль ядра системы играет не модель, а общая (интегрированная) база данных (ОБД), к которой могут обращаться различные проблемно-ориентированные модели, реализованные в форме программных приложений. Предполагается, что в ОБД хранятся информационные объекты (ИО), адекватно отображающие в информационном мире сущности физического мира — предметы, материалы, изделия, процессы и технологии, разнообразные документы, финансовые ресурсы, персонал подразделения и оборудование предприятия-изготовителя, условия эксплуатации изделия у заказчика, сервисная и ремонтная службы и т.д.

Указанные приложения обращаются в общую базу данных, находят в ней необходимые информационные объекты, обрабатывают их и помещают в ОБД результаты этой обработки.

В какой-то мере Н.Окино предвосхитил появление объектно-ориентированного подхода к созданию СУБД, предложив рассмат-

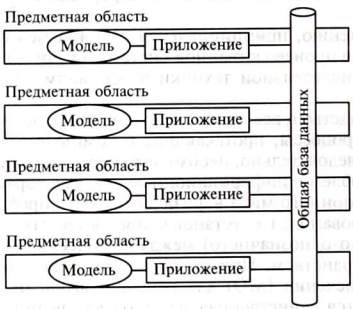


Рис. 18.1. Стратегия организации общей базы данных на основе локальных моделей

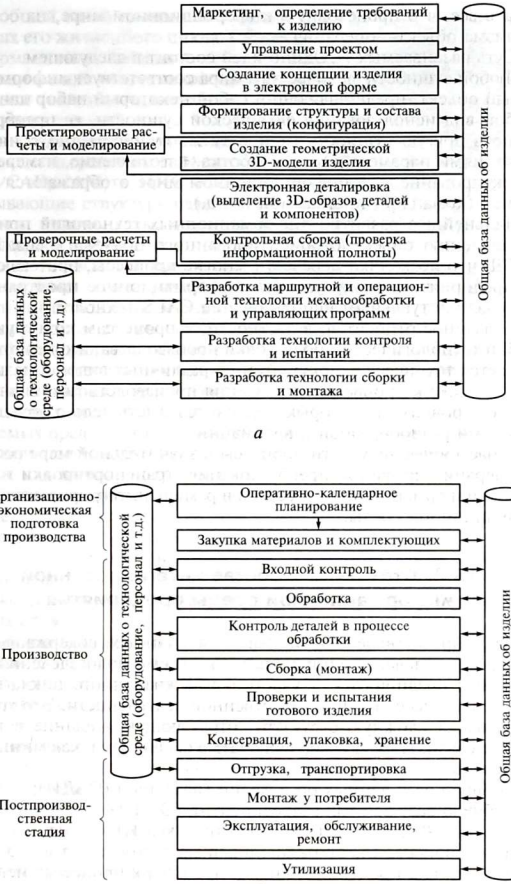


Рис. 18.2. Стратегия организации общей базы данных на основе многопользовательских (*а*) и распределенных (*б*) баз данных

ривать все, что происходит в информационном мире, на основе дуализма объект—операция.

Суть развиваемых Н. Окино идей состоит в следующем.

Любой сущности физического мира соответствует информационный объект, представляющий собой некоторый набор данных. Любой вид использования физической сущности, ее преобразование в другую сущность (или в ту же сущность, но с иными значениями параметров) — обработка, изготовление, измерение, проектирование — в информационном мире отображаются операцией (командой, программой и т.д.).

Дальнейшее развитие информационных технологий привело к появлению объектно-ориентированного подхода к созданию СУБД, что позволило перевести многие процессы, протекающие на предприятии, в виртуальное информационное пространство и сделало актуальным использование CALS-технологий.

Сказанное относится, в частности, к процессам конструкторской и технологической подготовки производства, в ходе которых создается техническая документация различных видов и назначений, а также к процессам управления производством и бизнесом на всех уровнях, на которых приходится иметь дело с большими объемами разнообразной информации.

В настоящее время эти процессы в значительной мере состоят из операций создания, преобразования, транспортировки и хранения информационных объектов в рамках интегрированной информационной среды.

18.2. Структура и состав интегрированной информационной среды предприятия

ИИС представляет собой хранилище данных, содержащее все сведения, создаваемые и используемые всеми подразделениями и службами предприятия — участниками жизненного цикла изделия — в процессе их производственной деятельности. Это хранилище имеет сложную структуру, отражающую внешние и внутренние связи. ИИС должна включать в свой состав, как минимум, две основные базы данных:

- общую базу данных об изделии (изделиях) (ОБДИ);
- общую базу данных о предприятии (ОБДП).

На рис. 18.2, где представлена структура ИИС во взаимодействии с процессами, протекающими на протяжении всего жизненного цикла продукции, видно, что в этих процессах используется информация, содержащаяся в ИИС, а информационные объекты, порождаемые в ходе процессов, возвращаются в ИИС для хранения и последующего использования в других процессах.

С общей базой данных об изделии связаны процессы на всех стадиях его жизненного цикла. Общая база данных о предприятии информационно связана с технологической и организационно-экономической подготовкой производства и собственно производством (включая процессы отгрузки и транспортировки готовой продукции).

При создании нового изделия и технологической подготовке его производства средствами конструкторских и технологических САПР (CAE/CAD/CAM) в ИИС создаются информационные объекты, описывающие структуру изделия, его состав и все входящие компоненты: детали, узлы, агрегаты, комплектующие, материалы и т.д. Каждый информационный объект обладает атрибутами, описывающими свойства физического объекта: технические требования и условия, геометрические (размерные) параметры, массогабаритные показатели, характеристики прочности, надежности, ресурса и другие его свойства и компоненты.

Информационные объекты в составе общей базы данных содержат в произвольном формате информацию, требуемую для выпуска и поддержки технической документации, необходимой на всех стадиях производственного процесса для всех изделий, выпускаемых предприятием. Каждый информационный объект идентифицируется уникальным кодом и может быть извлечен из базы данных для выполнения действий с ним. Общая база данных об изделиях обеспечивает информационное обслуживание и поддержку деятельности:

- заказчиков (владельцев) изделия;
- разработчиков (конструкторов), технологов, управленческого и производственного персонала предприятия-изготовителя;
- эксплуатационного, ремонтного персонала и специализированных служб.

Информационные объекты, входящие в ОБДИ (рис. 18.3), можно условно подразделить на три раздела:

- нормативно-справочный;
- долговременный;
- актуальный.

Содержание *нормативно-справочного раздела* ОБДИ обновляется по мере поступления новых и отмены действующих нормативных документов.

В *долговременном разделе* должны храниться ИО, содержащие данные, аккумулирующие собственный опыт предприятия.

Долговременный раздел ОБДИ дополняется и обновляется по мере появления новых технических решений, признанных типовыми и пригодными для дальнейшего использования.

В *актуальном разделе* (по-видимому, самом большом по объему и самом сложном по структуре) должны храниться ИО, содержащие данные об изделиях, находящихся на различных стадиях ЖЦ.

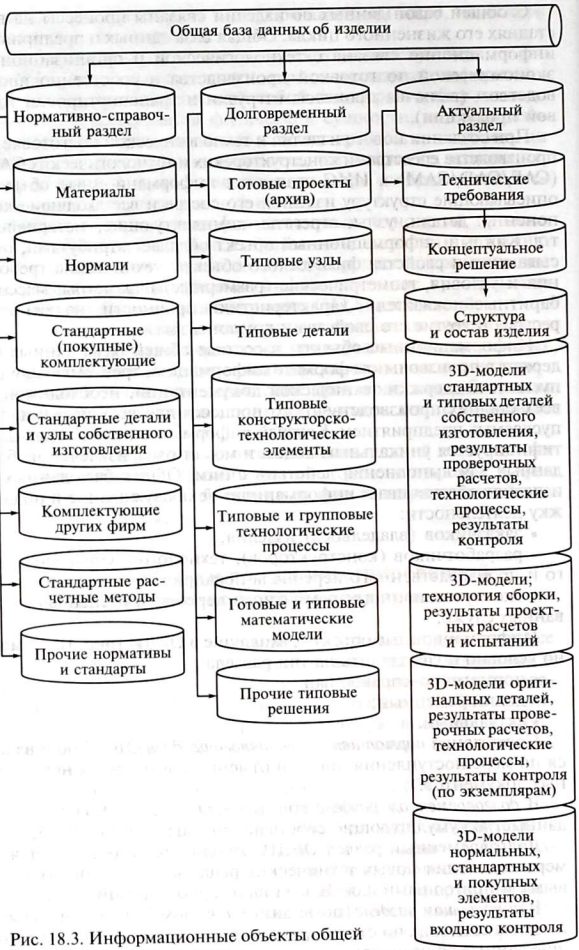


Рис. 18.3. Информационные объекты общей базы данных

Структура этого раздела очень приблизительна и требует развития и уточнения, в том числе разбивки на дополнительные подразделы (классификационные уровни).

Как уже отмечалось, кроме информационных объектов, относящихся (прямо или косвенно) к изделиям, в ИИС содержится информация о предприятии: о его производственной и управленческой структуре, технологическом и вспомогательном оборудовании, персонале, финансах и т.д. Вся совокупность этих данных образует ОБДП, которая также состоит из нескольких разделов: экономика и финансы, внешние связи предприятия, производственно-технологическая среда предприятия, система качества.

В разделе, посвященном экономике и финансам, должны храниться информационные объекты, содержащие сведения:

- о конъюнктуре рынка изделий предприятия, включая цены и их динамику;
- состоянии финансовых ресурсов предприятия;
- ситуации на фондовом и финансовом рынках (курсы акций предприятия, биржевые индексы, процентные ставки, валютные курсы и т.д.);
- реальном и прогнозируемом портфеле заказов;
- прочие сведения финансово-экономического и бухгалтерского характера.

В разделе, посвященном внешним связям предприятия, должны храниться ИО, содержащие сведения о фактических и возможных поставщиках и потребителях (заказчиках). Данный раздел формируется и используется в процессе маркетинговых исследований.

В разделе, посвященном производственно-технологической среде, должны храниться информационные объекты, содержащие сведения:

- о производственной структуре предприятия;
- технологическом, вспомогательном и контрольно-измерительном оборудовании;
- транспортно-складской системе предприятия;
- энерговооруженности предприятия;
- кадрах;
- прочие данные о предприятии.

В разделе, посвященном системе качества, должны храниться сведения:

- о структуре действующей на предприятии системы качества;
- действующих на предприятии стандартах по качеству;
- международных и российских стандартах по качеству;
- должностных инструкциях в области качества;
- прочая информация о системе качества.

Из ИИС могут быть извлечены разнообразные документы, необходимые для функционирования предприятия. Документы мо-

гут быть представлены как в электронном, так и в традиционном бумажном виде.

При выполнении электронного проектирования средствами систем CAE/CAD/CAM должны использоваться и электронные средства управления конфигурацией, отвечающие, в частности, требованиям стандарта ИСО 10303-203.

18.3. Управление интегрированной информационной средой предприятия

Управление интегрированной информационной средой предприятия предполагает, что все процессы, протекающие в ней, управляемы, т.е. поддаются воздействиям со стороны уполномоченных лиц (администраторов) и соответствующих программных средств. Совокупность таких средств принято называть системой управления базами данных. Функции СУБД следующие:

- помещение информации в базу данных;
- хранение информации, в том числе создание резервных копий;
- обновление данных (ввод новых данных взамен данных, утративших актуальность);
- обеспечение достоверности и целостности данных;
- поиск данных по различным признакам;
- создание отчетов;
- установление (изменение) и оперативная проверка прав доступа пользователей к данным и т.д.

Распределенный характер ИИС, в отличие от традиционных БД, требует создания специальной инфраструктуры, обеспечивающей накопление, хранение и передачу данных между всеми заинтересованными участниками ЖЦ изделия. Такая инфраструктура должна представлять собой комплекс программных и аппаратных средств, позволяющий решать перечисленные задачи.

В рамках традиционного предприятия, расположенного на единой (и единственной) производственной площадке, такая инфраструктура создается на основе локальной вычислительной сети и соответствующего системного и прикладного программного обеспечения.

Для предприятий с географически распределенной производственной структурой, особенно для виртуальных предприятий, эта проблема очень важна.

Анализ состояния телекоммуникационных средств и систем позволяет утверждать, что в настоящее время основой инфраструктуры виртуального предприятия, а также предприятия с географически распределенной структурой может служить глобальная сеть Интернет, в которой данные передаются с помощью про-

токола TCP/IP. Несмотря на внешнюю простоту и доступность сети Интернет использование ее в качестве структурообразующего средства связано с рядом специфических проблем.

Первая проблема заключается в том, что для эффективного накопления, хранения и использования данных всеми участниками информационного обмена в соответствии с технологиями ИПИ хранилище данных должно быть логически локализовано в форме, которую в Интернет-технологиях принято называть порталом. Иными словами, должен быть создан специальный узел сети Интернет, предназначенный для информационного обслуживания предприятия, виртуального предприятия или корпорации.

Вторая проблема связана с тем, что этот узел сети Интернет и соответственно участники информационного обмена должны быть ограждены от вмешательства посторонних лиц и организаций даже при отсутствии у них какого-либо злого умысла или враждебных интересов.

Третья проблема заключается в защите информации от несанкционированного доступа лиц и организаций, имеющих цель использовать эту информацию во враждебных целях (похищение сведений, составляющих государственную и (или) коммерческую тайну, нарушение целостности и (или) достоверности данных, передаваемых участниками информационного обмена и т.д.).

18.4. Управление качеством

Система управления качеством продукции является элементом управленческой деятельности предприятия. В соответствии с международным стандартом ИСО 9000:2000 управление качеством должно базироваться на информационной системе, поддерживающей автоматизированную обработку данных и документирование процессов обеспечения качества на всех стадиях ЖЦ изделия, а также автоматизированное управление этими процессами, данными и документацией. В этом смысле управление качеством становится неотъемлемой частью автоматизированной системы управления предприятием и может быть отнесена к CALS-технологиям. Это означает, что информация, циркулирующая в системе управления качеством, должна быть представлена в форматах, регламентированных стандартами CALS-технологий, и состоять из набора информационных объектов, входящих в ИИС предприятия.

Укрупненная структура системы управления качеством, приведенная на рис. 18.4, показывает связи системы с объектом управления (процессами ЖЦ продукции), а также с внешней по отношению к рассматриваемой системе средой, которую в данном случае представляет некий обобщенный потребитель со своими требованиями и степенью удовлетворенности продукцией.

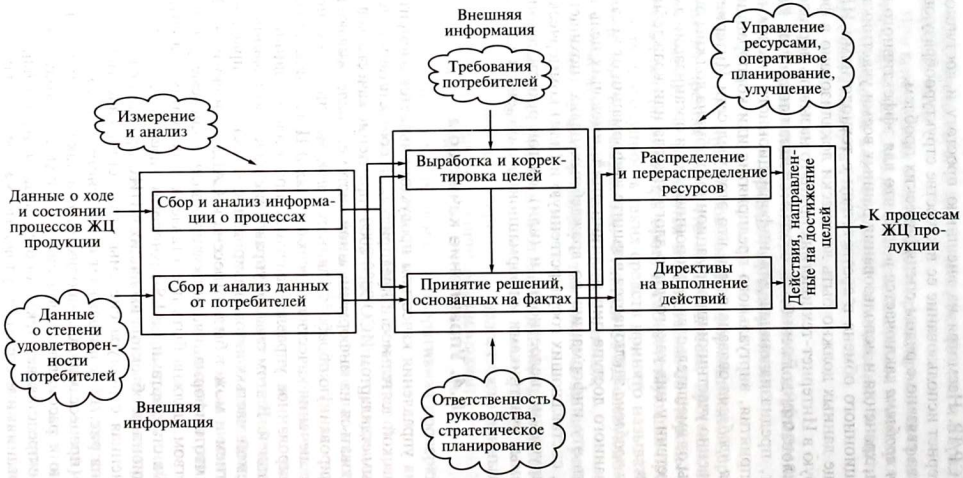


Рис. 18.4. Структура системы управления качеством

Присутствующие в структуре приведенной системы блоки выработки и корректировки целей и принятия решений, основанных на фактах, вместе эквивалентны тому, что в стандарте ИСО 9000 : 2000 называется ответственностью руководства и планированием (в данном случае — стратегическим). Блоки сбора и анализа данных от потребителей и информации о процессах отражают процессы, определенные в стандарте как измерение и анализ. Группа блоков, связанных с реализацией решений (распределение и перераспределение ресурсов, директивы на выполнение действий и сами действия, направленные на достижение целей), отражает процессы, определяемые в стандарте как управление ресурсами, планирование (в данном случае — оперативным) и улучшение.

Как уже отмечалось, при создании и технологической подготовке производства нового изделия средствами конструкторских и технологических САПР (CAD/CAM) в ИИС создаются ИО, описывающие структуру изделия, его состав и все входящие в него компоненты: детали, узлы, агрегаты, комплектующие, материалы и т.д. Каждый ИО обладает набором характеристик (атрибутов), описывающих свойства реального объекта, отображением которого он является. Для процесса управления качеством такими характеристиками являются технические требования и технические условия, которым должен удовлетворять реальный объект. Кроме информации об изделии в ИИС содержится информация о производственной среде предприятия, в составе которой находятся данные, относящиеся к управлению качеством.

18.5. Управление потоками работ

Понятие *поток работ* (workflow) — одно из базовых в современной практике управления и, в частности, в области ИПИ. Это понятие объединяет подходы к формализации и управлению бизнес-процессами предприятия, а также программные средства, реализующие эти подходы. Популярность и многообразие реализаций технологий workflow привели к созданию в середине 1990-х гг. международной ассоциации WfMC (Workflow Management Coalition — WfMC), занимающейся стандартами, регламентирующими требования к системам workflow и средствам их реализации.

Согласно глоссарию WfMC, бизнес-процесс — это одна или более связанных между собой процедур или операций (функций), которые совместно реализуют некую бизнес-задачу или политическую цель предприятия, как правило, в рамках организационной структуры, описывающей функциональные роли и отношения. Бизнес-процесс может целиком осуществляться в пределах одного организационного подразделения, охватывать несколько подразделений в рамках организации или даже несколько различ-

ных организаций, как, например, в системе отношений клиент — поставщик. Бизнес-процесс может включать в себя формальные и неформальные взаимодействия между участниками; его продолжительность может изменяться в широких пределах.

Поток работ — это упорядоченное во времени множество рабочих заданий, получаемых сотрудниками, которые обрабатывают эти задания вручную или с помощью средств механизации (автоматизации) в последовательности и в рамках правил, определенных для данного бизнес-процесса. Бизнес-процесс — это своего рода конвейер, работающий по своим правилам и технологиям, а поток работ (заданий) аналогичен потоку изделий (узлов, деталей), которые этот конвейер передвигает.

Бизнес-процесс объединяет поток заданий и функции, которые должны выполняться с элементами (заданиями) этого потока, людей и оборудование, которые реализуют эти функции, а также правила, управляющие последовательностью их выполнения. Технология workflow призвана все это автоматизировать.

Приведем два определения из глоссария WfMC.

1. *Поток работ* — полная или частичная автоматизация бизнес-процесса, при которой документы, информация или задания передаются для выполнения необходимых действий от одного участника к другому в соответствии с набором процедурных правил.

2. *Система управления потоком работ* — система, которая описывает этот поток (бизнес-процесс), создает его и управляет им с помощью программного обеспечения, которое способно интерпретировать описание процесса, взаимодействовать с его участниками и при необходимости вызывать соответствующие программные приложения и инструментальные средства.

Такая система автоматизирует процесс, а не функцию. Появление систем управления потоком работ и соответствующих программных средств — это реакция рынка информационных технологий на внедрение новых принципов управления предприятиями. Сегодня эти принципы эволюционируют от функциональной ориентации (придуманной Адамом Смитом еще в 1776 г. и успешно работающей на протяжении более двух столетий) в направлении процессной ориентации.

Практически все предыдущие решения (чаще всего реализованные в технологиях локальных СУБД) позволяли достаточно эффективно автоматизировать отдельные операции и функции, а не процесс. В рамках этих решений сотрудники, сидя за своими компьютерами (или терминалами), обмениваются информацией с базами данных и между собой, получают данные, справки, документы, формируют отчеты. При этом последовательность действий сотрудников и правила их взаимодействия определены в лучшем случае инструкциями, а за правильностью и сроками их

выполнения следит вышестоящее начальство. Информационная система все это никак не поддерживает.

Процессный подход заставил руководство предприятий сконцентрировать внимание на правилах взаимодействия участников процесса, так как именно взаимодействия в силу своей размытости и недостаточной определенности являются основным источником неоправданных издержек и потерь. Потребность в средствах автоматического отслеживания порядка и времени выполнения отдельных функций (операций), маршрутов документов, занятости сотрудников на различных стадиях процесса привели к созданию разнообразных систем управления потоками работ и к утверждению этого понятия в качестве одного из базовых вариантов концепции ИПИ.

Контрольные вопросы

1. Что означает термин *интегрированная информационная среда*?
2. Что означает термин *информационный объект*?
3. Какая информация должна содержаться в общей базе данных об изделии?
4. Какая информация должна содержаться в общей базе данных предприятия?
5. Какие задачи и в соответствии с каким стандартом решает система управления качеством?
6. Какая связь существует между понятиями *управление потоками работ* и *бизнес-процессы*?

СПИСОК ЛИТЕРАТУРЫ

1. *Буч Г.* Язык UML. Руководство пользователя : пер. с англ. / Г. Буч, Д. Рамбо, А. Джекобсон. — М. : ДМК, 2000. — 432 с.

2. *Карпова Т. С.* Базы данных : модели, разработка, реализация / Т. С. Карпова. — СПб. : Питер, 2001. — 304 с.

3. *Рамел Д.* Visual Basic.Net. Справочник программиста / Д. Рамел. — М. : Эком, 2002.

4. *Конноли Т.* Базы данных. Проектирование, реализация и сопровождение. Теория и практика : пер. с англ. / Т. Конноли, Б. Каролин. — 3-е изд. — М. : Изд. дом «Вильямс», 2003. — 1440 с.

5. *Маклаков С. В.* CASE-средства разработки информационных систем / С. В. Маклаков. — М. : Диалог-МИФИ, 2000. — 256 с.

6. *Матиас Н.* Знакомьтесь : World Wide Web : пер. с нем. / Н. Матиас. — Киев : Торгово-издательское бюро BHV, 1996. — 336 с.

7. Толковый словарь по вычислительным системам / [под ред. В. Иллингуорта и др. ; пер с англ. А. К. Белоцкого и др.] ; под ред. Е. К. Масловского. — М. : Машиностроение, 1990. — 560 с.

8. *Фуфаев Э. В.* Пакеты прикладных программ / Э. В. Фуфаев, Л. И. Фуфаева. — М. : Изд. центр «Академия», 2004. — 352 с.

9. *Фуфаев Э. В.* Базы данных / Э. В. Фуфаев, Д. Э. Фуфаев. — М. : Изд. центр «Академия», 2005. — 320 с.

ОГЛАВЛЕНИЕ

Учебное издание

Фуфаев Эдуард Валентинович,
Фуфаев Дмитрий Эдуардович

Разработка и эксплуатация удаленных баз данных

Учебник

4-е издание, стереотипное

Редактор *В. Н. Махова*
Технический редактор *Н. И. Горбачева*
Компьютерная верстка: *Р. Ю. Волкова*
Корректоры *Т. В. Кузьмина*, *Н. Л. Котелина*

Изд. № 104108336. Подписано в печать 09.09.2013. Формат 60×90/16.
Бумага офс. № 1. Гарнитура «Таймс». Печать офсетная. Усл. печ. л. 16,0.
Тираж 1 000 экз. Заказ № 34917.

ООО «Издательский центр «Академия». www.academia-moscow.ru
129085, Москва, пр-т Мира, 101В, стр. 1.
Тел./факс: (495) 648-0507, 616-00-29.
Санитарно-эпидемиологическое заключение № РОСС RU. АЕ51. Н 16476 от 05.04.2013.

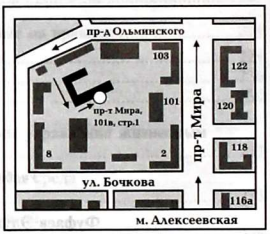
Отпечатано в соответствии с качеством предоставленных издательством
электронных носителей в ОАО «Саратовский полиграфкомбинат».
410004, г. Саратов, ул. Чернышевского, 59. www.sarpk.ru

Издательский центр «Академия»

Учебная литература для профессионального образования

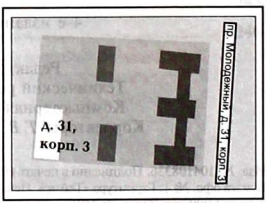
Наши книги можно приобрести (оптом и в розницу)

Москва
129085, Москва, пр-т Мира, д.101в, стр. 1
(м. Алексеевская)
Тел.: (495) 648-0507, факс: (495) 616-0029
E-mail: sale@academia-moscow.ru

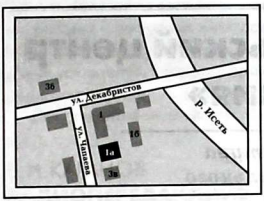


Филиалы:

Северо-Западный
194044, Санкт-Петербург,
ул. Чугунная, д. 14, оф. 319
Тел./факс: (812) 244-92-53
E-mail: spboffice@academizdat.ru

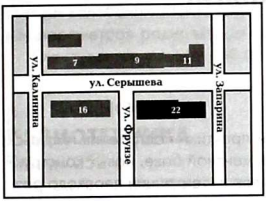


Приволжский
603101, Нижний Новгород,
пр. Молодежный, д. 31, корп. 3
Тел./факс: (831) 259-7431,
259-7432, 259-7433
E-mail: pf-academia@bk.ru

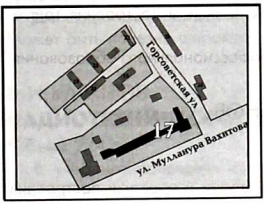


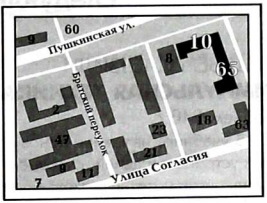
Уральский
620142, Екатеринбург, ул. Чапаева,
д. 1а, оф. 12а
Тел.: (343) 257-1006
Факс: (343) 257-3473
E-mail: academia-ural@mail.ru

Сибирский
630108, Новосибирск,
ул. Добролюбова, д. 31, корп. 4, а/я 73
Тел./факс: (383) 362-2145, 362-2146
E-mail: academia_sibir@mail.ru



Дальневосточный
680038, Хабаровск, ул. Серышева,
д. 22, оф. 519, 520, 523
Тел./факс: (4212) 56-8810
E-mail: filialdv-academia@yandex.ru





Южный
344082, Ростов-на-Дону,
ул. Пушкинская, д.10/65
Тел.: (863) 203-5512
Факс: (863) 269-5365
E-mail: academia-UG@mail.ru

Представительства:

в Республике Татарстан
420034, Казань, ул. Горсоветская,
д. 17/1, офис 36
Тел./факс: (843) 562-1045
E-mail: academia-kazan@mail.ru

в Республике Дагестан
Тел.: 8-928-982-9248

аналоговых и цифровых интегральных микросхем. Рассмотрены принципы построения функциональных электронных устройств на основе операционных усилителей, компараторов, таймеров, оптронов и других оптоэлектронных устройств.

Для студентов учреждений среднего профессионального образования.

В. И. КАГАНОВ

РАДИОПЕРЕДАЮЩИЕ УСТРОЙСТВА

Объем 288 с.

В учебнике изложены основы теории работы радиопередающих устройств, принципы их расчета и проектирования, применение в различных радиоэлектронных системах. Рассмотрены физические процессы, связанные с генерированием, усилением и модуляцией высоко- и сверхвысокочастотных колебаний. Освещены вопросы регулировки, испытаний и измерения параметров радиопередающих устройств.

Для студентов учреждений среднего профессионального образования.

В. И. КАГАНОВ

РАДИОТЕХНИКА

Объем 352 с.

В учебном пособии изложены принципы функционирования систем радиосвязи, радиовещания, радиолокации, радионавигации, радиоуправления, телевидения. Рассмотрены основы работы электронных приборов, радиопередатчиков, радиоприемников и СВЧ устройств. Приведены сведения из истории радиотехники, рассказано об ученых, изобретателях и инженерах, внесших наиболее весомый вклад в развитие этой науки.

Для студентов учреждений среднего профессионального образования. Может быть полезно широкому кругу читателей.

В. И. КАГАНОВ

РАДИОТЕХНИЧЕСКИЕ ЦЕПИ И СИГНАЛЫ

Объем 224 с.

В учебнике приведены основные сведения по передаче и приему сообщений с помощью радиосигналов и построению систем радиосвязи. Рассмотрены основы спектральной теории сигналов и их генерирование, преобразование, суммирование, модуляция, детектирование, демодуляция и обработка. Изложены основы теории радиоэлектронных ли-

нейных, нелинейных и параметрических цепей сосредоточенного и распределенного типа: их назначение, классификация, параметры, характеристики и расчет, в том числе и с помощью компьютера. Представлено большое число примеров по анализу радиоэлектронных цепей различного назначения.

Для студентов учреждений среднего профессионального образования.

И. М. МЫШЛЯЕВА

ЦИФРОВАЯ СХЕМОТЕХНИКА

Объем 400 с.

В учебнике рассмотрены арифметические основы теории цифровых устройств и основы алгебры логики, комбинационные цифровые устройства, последовательностные цифровые устройства, структура построения вычислительных устройств, запоминающие устройства, многопроцессорные устройства.

Для студентов учреждений среднего профессионального образования.

Е. И. НЕФЕДОВ

АНТЕННО-ФИДЕРНЫЕ УСТРОЙСТВА И РАСПРОСТРАНЕНИЕ РАДИОВОЛН

Объем 320 с.

В учебнике кратко изложены основные законы высокочастотной электродинамики, описаны наиболее применяемые на практике типы антенных устройств и фидеров (волноводов), рассмотрены особенности распространения радиоволн разных диапазонов. Часть материала посвящена новым классам интегральных линий передачи, полосковым и щелевым антеннам и антенным решеткам на их основе, а также некоторым базовым элементам антенно-фидерного тракта, построенным на плоскостных и объемных интегральных схемах СВЧ и КВЧ.

Для студентов учреждений среднего профессионального образования.

РАЗРАБОТКА И ЭКСПЛУАТАЦИЯ УДАЛЕННЫХ БАЗ ДАННЫХ

ISBN 978-5-4468-0467-2

Издательский центр
«Академия»
www.academia-moscow.ru