ФГБОУ ВПО «Воронежский государственный
технический университет»

Т.И. Сергеева М.Ю. Сергеев

РАСПРЕДЕЛЕННАЯ ОБРАБОТКА ДАННЫХ

Утверждено Редакционно-издательским советом университета
в качестве учебного пособия

Воронеж 2014

УДК 681.3

Сергеева Т.И. Распределенная обработка данных: учеб. пособие [Электронный ресурс]. – Электрон. текстовые и граф. данные (0,8 Мб) / Т.И. Сергеева, М.Ю. Сергеев. - Воронеж: ФГБОУ ВПО «Воронежский государственный технический университет», 2014. – 1 электрон. опт. диск (CD-ROM) : цв. – Систем. требования : ПК 500 и выше ; 256 Мб ОЗУ ; Windows XP ; SVGA с разрешением 1024x768 ; MS Word 2007 или более поздняя версия ; CD-ROM дисковод ; мышь. – Загл. с экрана. – Диск и сопровод. материал помещены в контейнер 12x14 см.

В учебном пособии рассматриваются особенности построения, функционирования и применения систем распределенной обработки данных. В пособии также описаны технологии создания баз данных и таблиц средствами СУБД MS SQL Server и Firebird. Описана реализация доступа к базам данных из приложений, созданных в Access 2007 и в среде программирования Delphi 7.0. Пособие содержит теоретические сведения и примеры написания запросов на языке SQL.

Издание соответствует требованиям Федерального государственного образовательного стандарта высшего профессионального образования по направлению 230100.68 «Информатика и вычислительная техника» (магистерская программа подготовки «Распределенные автоматизированные системы»), дисциплине «Распределенная обработка данных».

Табл. 3. Ил. 26. Библиогр.: 8 назв.

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

Рецензенты: кафедра вычислительной математики и прикладных информационных технологий Воронежского государственного университета (зав. кафедрой д-р техн. наук, проф. Т.М. Леденева);
д-р техн. наук, проф. В.Ф. Барабанов

ВВЕДЕНИЕ

Распределенная обработка данных может базироваться на централизованной базе данных и распределенных приложениях, работающих с ней, а также на использовании распределенных баз данных (РБД), находящихся в различных узлах сети.

В первой главе рассматриваются общие вопросы построения, функционирования и применения систем распределенной обработки данных. Рассматриваются модели архитектуры клиент-сервер, модели серверов баз данных, трехзвенные модели организации данных. Особое внимание уделено особенностям разработки и эксплуатации распределенных баз данных, управлению РБД.

Вторая глава содержит описание технологии создания баз данных и таблиц средствами СУБД SQL Server, имеющиеся типы данных, установку связей между таблицами, определение прав доступа к таблицам базы данных. В данной главе описывается также реализация доступа к таблицам базы данных из приложений, созданных в Access.

Третья глава пособия рассматривает особенности создания баз данных и таблиц в Firebird 2.0, используемые типы данных, установку связей между таблицами. Рассматривается реализация доступа к таблицам базы данных из приложений, созданных в среде Delphi.

Четвертая глава пособия посвящена языку структурированных запросов SQL. В данной главе рассматриваются следующие вопросы: общая структура запроса на выборку данных, запросы с вычисляемыми полями, с групповыми вычислениями, параметрические и перекрестные запросы. Приведено достаточное количество примеров написания запросов.

Пособие соответствует типовой программе по дисциплине «Распределенная обработка данных» и предназначено для магистров, обучающихся по программе «Распределенные автоматизированные системы».

1. ОБЩАЯ ХАРАКТЕРИСТИКА РАСПРЕДЕЛЕННЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

1.1. Режимы использования баз данных

При размещении базы данных (БД) на персональном компьютере, который не находится в сети, БД всегда используется в монопольном режиме. При использовании такой БД несколькими пользователями они могут работать с ней только последовательно. Корректная модификации БД в этом случае осуществляется организационными мерами — то есть определением требуемой последовательности работы конкретных пользователей с соответствующей БД.

Однако работа на изолированном компьютере с небольшой базой данных не является характерной для большинства приложений. БД отражает информационную модель реальной предметной области, она растет по объему и резко увеличивается количество задач, решаемых с ее использованием, и в соответствии с этим увеличивается количество приложений, работающих с единой базой данных. Компьютеры объединяются в локальные сети, и необходимость распределения приложений, работающих с единой базой данных по сети, постоянно растет.

Параллельный доступ к одной БД нескольких пользователей в том случае, если БД расположена на одной машине, соответствует режиму распределенного доступа к централизованной БД. Такие системы называются **системами распределенной обработки данных**.

Если же БД распределена по нескольким компьютерам, расположенным в сети, и к ней возможен параллельный доступ нескольких пользователей, то мы имеем дело с параллельным доступом к распределенной БД. Подобные системы называются **системами распределенных баз данных**. В общем случае режимы использования БД можно представить в следующем виде (см. рис. 1.1).

Таким образом, существует два основных варианта организации базы данных в локальной сети.

Первый вариант – **системы распределенной обработки данных**. Централизованная БД расположена на одной машине (сервере). К ней осуществляется параллельный доступ нескольких пользователей и приложений, находящихся на рабочих станциях, объединенных в вычислительную сеть. Централизованная организация данных позволяет облегчить обеспечение безопасности, целостности и непротиворечивости данных.

Второй вариант – **системы распределенных баз данных**. БД распределена на нескольких компьютерах, объединенных в сеть. К БД возможен параллельный доступ нескольких пользователей и приложений, расположенных в узлах сети.

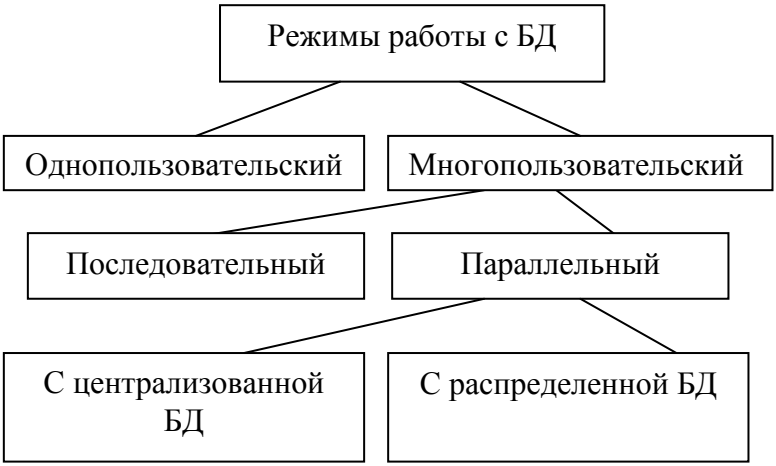

Рис. 1.1. Режимы работы с базой данных

Централизованная организация хранения данных имеет следующие особенности:

- большой объем обмена данными (высокий трафик);
- снижение надежности обмена данными;
- снижение общей производительности;

- централизованное управление данными;

- эффективные и проверенные методы обеспечения безопасности, целостности и непротиворечивости данных

Децентрализованное хранение данных, размещенных в узлах сети, имеет следующие характеристики:

- параллельная обработка данных и распределение нагрузки между узлами сети;

- повышение эффективности обработки данных при выполнении удаленных запросов;

- уменьшение временных затрат на обработку данных;

- усложнение процедур управления данными;

- усложнение поддержки безопасности, целостности и непротиворечивости данных.

1.2. Модели архитектуры клиент-сервер

При построении информационных систем (ИС), работающих с централизованной БД, широко используется архитектура клиент-сервер. Ее основу составляет организация взаимодействия клиента и сервера при управлении БД.

Архитектура клиент-сервер – структура информационной системы, в которой применено распределенное управление сервером и рабочими станциями (клиентами) для максимально эффективного использования вычислительных мощностей.

Сервер – узловая станция компьютерной сети, предназначенная для хранения и управления данными коллективного пользования и для обработки запросов, поступающих от пользователей других узлов.

Клиент – компьютер, обращающийся к совместно используемым ресурсам, которые представляются другим компьютером (сервером).

Структура распределенной ИС, построенной по архитектуре клиент-сервер с использованием сервера баз данных, показана на рис. 1.2.

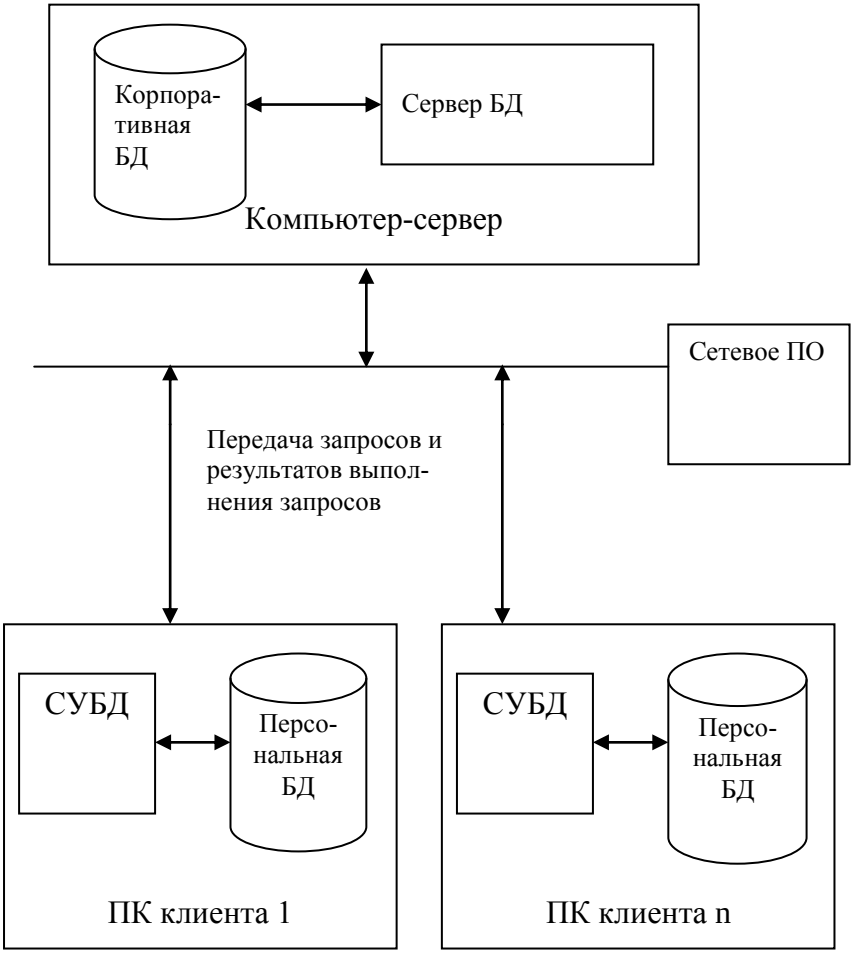

Рис. 1.2. Реализация клиент-серверной архитектуры

При такой архитектуре сервер баз данных обеспечивает выполнение основного объема обработки данных. Формируемые пользователем или приложением запросы поступают к серверу БД в виде инструкций языка SQL. Сервер баз данных выполняет поиск и извлечение нужных данных, которые затем пе-

редаются на компьютер пользователя. Преимуществом такого подхода является заметно меньший объем передаваемых данных.

Для создания и управления персональными БД и приложениями, работающими с ними, используются СУБД, такие как Access и Visual FoxPro фирмы Microsoft, Paradox фирмы Borland.

Корпоративная БД создается, поддерживается и функционирует под управлением сервера БД, например, Microsoft SQL Server, Oracle Server, InterBase (FireBird), Sybase, Informix.

В зависимости от размеров организации и особенностей решаемых задач информационная система может иметь одну из следующих конфигураций:

- компьютер-сервер, содержащий корпоративную и персональные базы;
- компьютер-сервер и персональные компьютеры с ПБД;
- несколько компьютеров-серверов и персональных компьютеров с ПБД.

Использование архитектуры клиент-сервер дает возможность постепенного наращивания информационной системы предприятия, во-первых, по мере развития предприятия; во-вторых, по мере развития самой информационной системы.

Разделение общей БД на корпоративную БД и персональные БД позволяет уменьшить сложность проектирования БД по сравнению с централизованным вариантом, а значит снизить вероятность ошибок при проектировании и стоимость проектирования.

Согласно Эталонной модели Архитектуры открытых систем OSI функция управления БД относится к прикладному уровню.

СУБД как программа, поддерживающая интерфейс с пользователем, реализует следующие основные функции:

- управление данными, находящимися в базе;
- обработка информации с помощью прикладных программ;

- представление информации в удобном для пользователя виде.

При размещении СУБД в сети возможны различные варианты распределения функций по узлам сети. В зависимости от числа узлов сети, между которыми выполняется распределение функций СУБД, можно выделить двухзвенные и трехзвенные модели. Место разрыва функций соединяется коммуникационными элементами (средой передачи информации в сети).

Двухзвенные модели соответствуют распределению функций СУБД между двумя узлами сети. Компьютер (узел сети), на котором обязательно присутствует функция управления данными, называют компьютером-сервером. Компьютер, близкий к пользователю и обязательно занимающийся вопросами представления информации, называют компьютером-клиентом.

Наиболее типичными вариантами разделения функций между компьютером-сервером и компьютером-клиентом являются следующие [1].

1. **Распределенное представление**:

- компьютер-сервер – управление данными, обработка, представление;
- компьютер-клиент – представление.

2. **Удаленное представление**:

- компьютер-сервер – управление данными, обработка;
- компьютер-клиент – представление.

3. **Распределенная функция**:

- компьютер-сервер – управление данными, обработка;
- компьютер-клиент – обработка, представление.

4. **Удаленный доступ к данным**

- компьютер-сервер – управление данными;
- компьютер-клиент – обработка, представление.

5. **Распределенная БД**

- компьютер-сервер – управление данными;
- компьютер-клиент – управление данными, обработка, представление.

Перечисленные способы распределения функций в системах с архитектурой клиент-сервер иллюстрируют различные варианты организации информационных систем: от мощного сервера, когда практически вся работа производится на нем, до мощного клиента, когда большая часть функций выполняется на рабочей станции, а сервер обрабатывает поступившие к нему по сети SQL-вызовы.

В моделях удаленного доступа к данным и удаленного представления производится строгое распределение функций между компьютером–клиентом и компьютером-сервером.

В других моделях имеет место выполнение одной из функций одновременно на двух компьютерах: управление данными (модель распределенной БД), обработка информации (модель распределенной функции), представление информации (модель распределенного представления).

Наиболее распространенными являются модели удаленного доступа к данным и модели удаленного представления (или модели сервера БД).

В модели **удаленного доступа к данным** (Remote Data Access – RDA) программы, реализующие функции представления информации и обработку данных, совмещены и выполняются на компьютере-клиенте.

Управление данными сосредоточено на компьютере-сервере, к которому обращается приложение, расположенное на компьютере-клиенте, с помощью языка запросов SQL или с помощью функций специальной библиотеки API (Application Programming Interface – интерфейс прикладного программирования).

Основное достоинство RDA-модели состоит в большом количестве СУБД, имеющих развитые инструментальные средства для создания программ клиентской части.

Средства разработки чаще всего поддерживают графический интерфейс пользователя в MS Windows, стандарт интерфейса ODBC и средства автоматической генерации кода.

RDA-модель имеет следующие недостатки:

- довольно высокая загрузка системы передачи данных, так как обработка данных реализуется в приложении, а обрабатываемые данные расположены на удаленном сервере;

- сложность модификации и сопровождения; в приложениях функции обработки и представления тесно связаны, при изменении выполняемых функций требуется переделка всей прикладной части, что усложняет модификацию систем.

Модель **сервера БД** – модель **удаленного представления** (DataBase Server – DBS) отличается тем, что функции компьютера-клиента ограничиваются функциями представления информации, а функции обработки данных обеспечиваются приложением, расположенным на компьютере-сервере. Эта модель является более технологичной чем RDA-модель и применяется в таких СУБД как Sybase, Oracle, Ingress. Приложения реализуются в виде хранимых процедур, которые хранятся в словаре БД.

Достоинства модели DBS следующие:

- возможность централизованного администрирования приложений на этапах разработки, сопровождения и модификации;

- эффективное использование вычислительных и коммуникационных ресурсов в связи с меньшими затратами на пересылку данных в сети.

Недостатки модели DBS следующие.

Ограниченность средств разработки хранимых процедур. Сильная привязка операторов разработки хранимых процедур к конкретной СУБД. Язык написания хранимых процедур является процедурным расширением языка SQL, не имеет функциональных возможностей традиционных языков программирования. В большинстве СУБД нет удовлетворительных средств отладки и тестирования хранимых процедур.

Низкая эффективность использования вычислительных ресурсов ЭВМ, так как не удается организовать эффективное управление входным потоком запросов к программам компью-

тера-сервера, а также обеспечить перемещение процедур на другие компьютеры-серверы.

Модель **распределенного представления** имеет мощный компьютер-сервер, а клиентская часть системы практически вырождена. Клиентская часть просто отображает информацию на экране монитора.

СУБД подобного рода используют в сетях, поддерживающих работу так называемых X-терминалов. В таких системах основной компьютер (хост-машина) должен иметь достаточную мощность, чтобы обслуживать несколько X-терминалов. X-терминал тоже должен обладать достаточно быстрым процессором и иметь достаточный объем оперативной памяти. Все программное обеспечение находится на хост-машине. Программное обеспечение X-терминала, выполняющее функции управления представлением и сетевые функции, загружается по сети с сервера при включении X-терминала.

Модель распределенного представления имели СУБД ранних поколений. В роли X-терминалов выступали дисплейные станции и абонентские пункты (локальные и удаленные).

По модели распределенного представления построены системы обслуживания пользователей БД в гетерогенной (неоднородной) среде. Серверная часть таких систем обычно обеспечивает некоторый унифицированный интерфейс, а клиентские части реализуют функции учета специфики конечного оборудования или преобразования одного формата представления информации в другой.

Модель распределенного представления реализует централизованную схему управления вычислительными ресурсами. Это обеспечивает основное преимущество данной модели – простота обслуживания и управления доступом к системе и относительная дешевизна (из-за невысокой стоимости терминалов).

Недостатки данной модели следующие: ненадежность системы при невысокой надежности центрального узла; высо-

кие требования к серверу по производительности при большом числе клиентов.

В модели **распределенной функции** обработка данных распределена по двум узлам. Такую модель могут иметь информационные системы, в которых общая часть прикладных функций реализована на компьютере-сервере, а специфические функции обработки информации выполняются на компьютере-клиенте. Функции общего характера могут обеспечивать целостность данных и реализовываться в виде хранимых процедур. Прикладные функции реализуют специальную обработку данных и включены в приложение, выполняющееся на компьютере-клиенте. Подобную модель также могут иметь ИС, использующие информацию из нескольких неоднородных БД.

Модель **распределенной БД** предполагает использование мощного компьютера-клиента. В данной модели данные хранятся на компьютере-сервере и компьютере-клиенте.

Возможны два варианта взаимосвязи баз данных:

- в локальной и удаленной базах хранятся отдельные части единой БД;

- локальная и удаленная БД являются синхронизируемыми друг с другом копиями.

Достоинства модели распределенной БД следующие:

- гибкость создаваемых на ее основе ИС, позволяющих компьютеру-клиенту обрабатывать локальные и удаленные БД;

- высокая живучесть, разрыв соединения сервера и клиента не приводит к краху системы, работа которой возобновляется с восстановлением соединения.

К недостатку модели можно отнести высокие затраты при выполнении большого числа одинаковых приложений на компьютерах-клиентах.

Существует второй вариант разделения функций, которые выполняются при хранении и обработке данных в информационной системе. Применение технологии «клиент-сервер» применительно к технологии баз данных предлагает разделить

функции стандартного интерактивного приложения на пять основных групп [3]:

– 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, приведена на рис. 1.3.

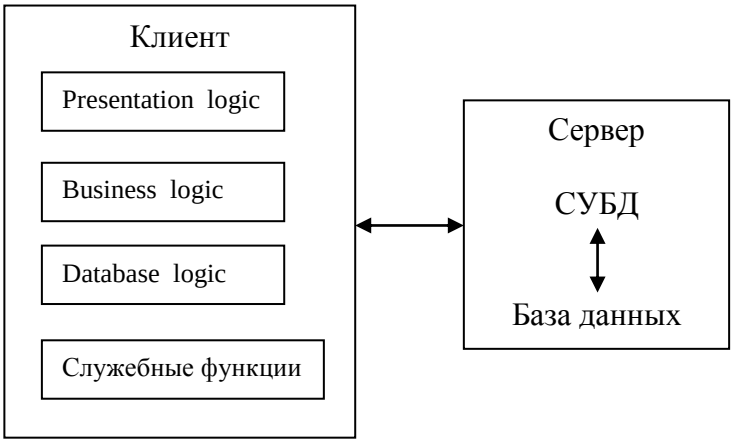

Рис. 1.3. Структура типового интерактивного приложения, работающего с базой данных

Презентационная логика (Presentation Logic) - это часть приложения, которая отображает результаты работы приложения, и реализует диалог с пользователем. Сюда относятся все интерфейсные экранные формы, которые пользователь видит

или заполняет в ходе работы приложения, к этой же части относится все то, что выводится пользователю на экран как результаты решения некоторых промежуточных задач либо как справочная информация. Поэтому 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 задач приложения. Обычно этот код пишется с использованием различных языков программирования, таких как C, C++, Pascal, Visual Basic и др.

Логика обработки данных (Data manipulation Logic) – это часть кода приложения, которая связана с обработкой данных внутри приложения. Данными управляет собственно СУБД. Для обеспечения доступа к данным используются язык запросов и средства манипулирования данными стандартного языка 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 для рассмотрения архитектуры приложения надо их выделить в отдельную часть приложения.

В централизованной архитектуре (Host-based processing) эти части приложения располагаются в единой среде и комбинируются внутри одной исполняемой программы.

В децентрализованной архитектуре эти задачи могут быть по-разному распределены между серверным и клиентским процессами. В зависимости от характера распределения можно выделить следующие модели распределений [2]:

– распределенная презентация (Distribution presentation, DP);

– удаленная презентация (Remote Presentation, RP);

– распределенная бизнес-логика (Remote business logic, RBL);

– распределенное управление данными (Distributed data management, DDM);

– удаленное управление данными (Remote data management, RDA).

Эта условная классификация показывает, как могут быть распределены отдельные задачи между серверным и клиентскими процессами. В этой классификации отсутствует реализация удаленной бизнес-логики. Действительно, считается, что она не может быть удалена сама по себе полностью. Считается, что она может быть распределена между разными процессами, которые в общем-то могут выполняться на разных платформах, но должны корректно кооперироваться (взаимодействовать) друг с другом.

В дальнейшем более подробно рассматриваются наиболее часто используемые модели распределения функций между сервером и клиентом.

Модель удаленного управления данными. Модель файлового сервера.

Модель удаленного управления данными также называется моделью файлового сервера (File Server, FS). В этой модели презентационная логика и бизнес-логика располагаются на клиенте. На сервере располагаются файлы с данными и поддерживается доступ к файлам. Функции управления информационными ресурсами в этой модели находятся на клиенте.

Распределение функций в этой модели представлено на рис. 1.4.

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, находится на клиенте.

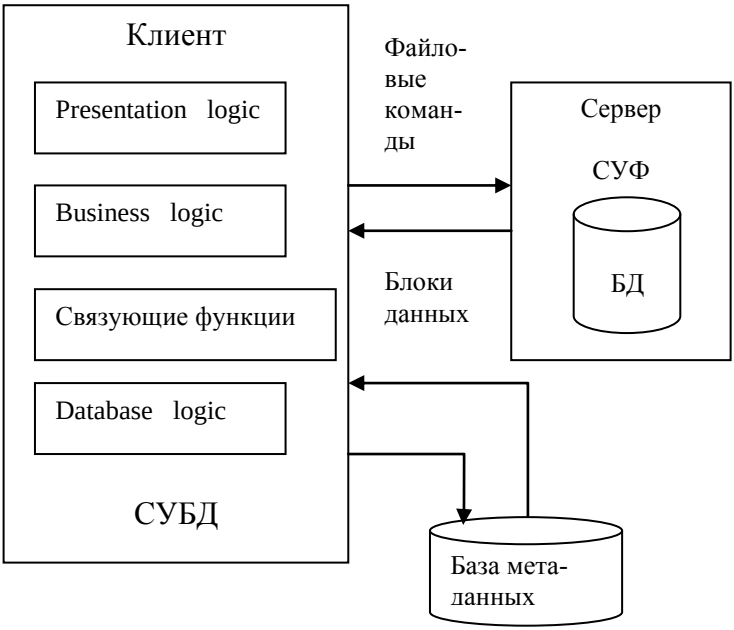

Рис. 1.4. Модель файлового сервера

Достоинства этой модели в том, что имеется разделение монопольного приложения на два взаимодействующих процесса. При этом сервер (серверный процесс) может обслуживать множество клиентов, которые обращаются к нему с запросами. Собственно СУБД должна находиться в этой модели на клиенте.

Запрос клиента формулируется в командах языка манипулирования данных. СУБД переводит этот запрос в последовательность файловых команд. Каждая файловая команда вызывает перекачку блока информации на компьютер клиента, далее на компьютере-клиенте СУБД анализирует полученную информацию, и если в полученном блоке не содержится ответ на запрос, то принимается решение о перекачке следующего блока информации и т. д.

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

Недостатки данной организации следующие:

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

Модель удаленного доступа к данным. В модели удаленного доступа (Remote Data Access, RDA) база данных хранится на сервере. На сервере же находится ядро СУБД.

На клиенте располагается презентационная логика и бизнес-логика приложения. Клиент обращается к серверу с запросами на языке SQL.

Структура модели удаленного доступа приведена на рис. 1.5.

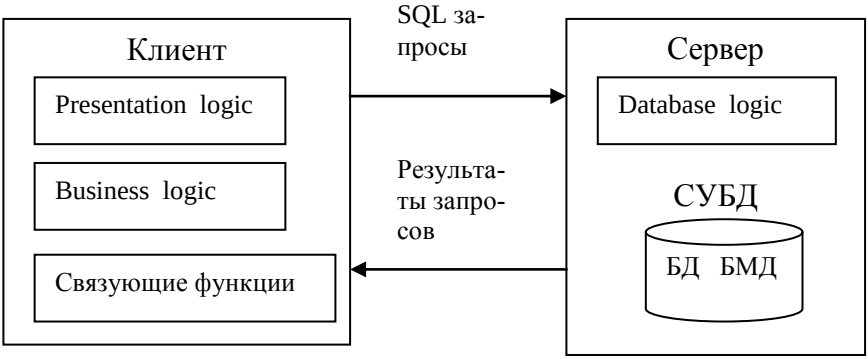

Рис. 1.5. Модель удаленного доступа (RDA)

Преимущества данной модели:

– перенос компонента представления и прикладного компонента на клиентский компьютер существенно разгрузил сер-

вер БД, сводя к минимуму общее число процессов в операционной системе;

– сервер БД освобождается от несвойственных ему функций; 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 SQL, и их объем существенно меньше. В ответ на запросы клиент получает только данные, релевантные запросу, а не блоки файлов, как в FS-модели.

Основное достоинство RDA-модели – унификация интерфейса «клиент-сервер», стандартом при общении приложения-клиента и сервера становится язык SQL.

Недостатки:

– все-таки запросы на языке SQL при интенсивной работе клиентских приложений могут существенно загрузить сеть;

– так как в этой модели на клиенте располагается и презентационная логика, и бизнес-логика приложения, то при повторении аналогичных функций в разных приложениях код соответствующей бизнес-логики должен быть повторен для каждого клиентского приложения. Это вызывает излишнее дублирование кода приложений;

– сервер в этой модели играет пассивную роль, поэтому функции управления информационными ресурсами должны выполняться на клиенте. Действительно, например, если необходимо выполнять контроль страховых запасов товаров на складе, то каждое приложение, которое связано, с изменением состояния склада, после выполнения операций модификации данных, имитирующих продажу или удаление товара со склада, должно выполнять проверку на объем остатка, и в случае, если он меньше страхового запаса, формировать соответствующую заявку на поставку требуемого товара. Это усложняет клиентское приложение, с одной стороны, а с другой — может вызвать необоснованный заказ дополнительных товаров несколькими приложениями.

Модель сервера баз данных. Для того чтобы избавиться от недостатков модели удаленного доступа, должны быть соблюдены следующие условия.

1. Необходимо, чтобы БД в каждый момент времени отражала текущее состояние предметной области, которое определяется не только собственно данными, но и связями между объектами данных. То есть данные, которые хранятся в БД, в каждый момент времени должны быть непротиворечивыми.

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

3. Необходим постоянный контроль 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.

4. Необходимо, чтобы возникновение некоторой ситуации в БД четко и оперативно влияло на ход выполнения прикладной задачи.

5. Одной из важнейших проблем СУБД является контроль типов данных. В настоящий момент СУБД контролирует синтаксически только стандартно-допустимые типы данных, то есть такие, которые определены в DDL (data definition language) — языке описания данных, который является частью SQL. Однако в реальных предметных областях действуют данные, которые несут в себе еще и семантическую составляющую, например, это координаты объектов или единицы различных метрик,

например рабочая неделя в отличие от реальной имеет сразу после пятницы понедельник.

Данную модель поддерживают большинство современных СУБД: Informix, Ingres, Sybase, Oracle, MS SQL Server. Основу данной модели составляет механизм хранимых процедур как средство программирования процедур обработки данных на сервере, механизм триггеров как механизм отслеживания текущего состояния информационного хранилища и механизм ограничений на пользовательские типы данных, который иногда называется механизмом поддержки доменной структуры. Модель сервера баз данных представлена на рис. 1.6.

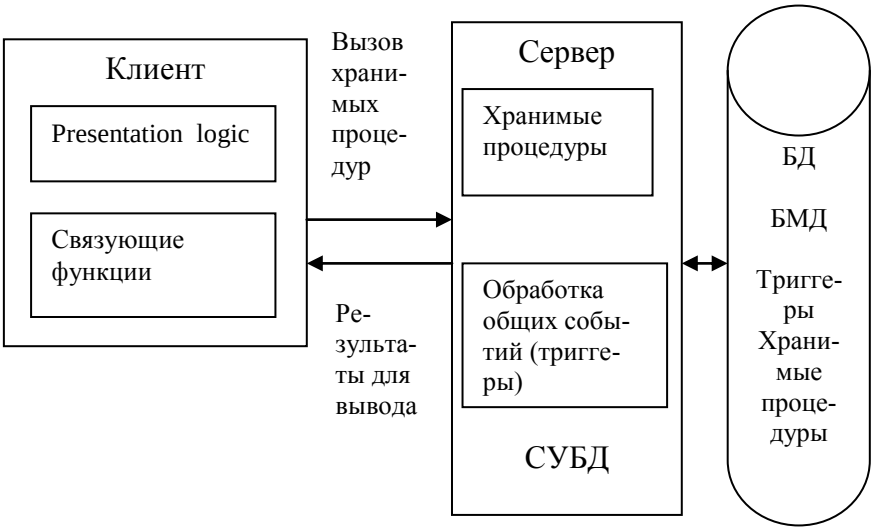

Рис. 1.6. Модель активного сервера БД

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 регист-

рирует все изменения в БД, которые в ней предусмотрены. Сервер возвращает клиенту данные, релевантные его запросу, которые требуются клиенту либо для вывода на экран, либо для выполнения части бизнес-логики, которая расположена на клиенте. Трафик обмена информацией между клиентом и сервером резко уменьшается.

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

Термин «триггер» взят из электроники и семантически очень точно характеризует механизм отслеживания специальных событий, которые связаны с состоянием БД. Триггер в БД является как бы некоторым тумблером, который срабатывает при возникновении определенного события в БД. Ядро СУБД проводит мониторинг всех событий, которые вызывают созданные и описанные триггеры в БД, и при возникновении соответствующего события сервер запускает соответствующий триггер. Каждый триггер представляет собой также некоторую программу, которая выполняется над базой данных. Триггеры могут вызывать хранимые процедуры.

Механизм использования триггеров предполагает, что при срабатывании одного триггера могут возникнуть события, которые вызовут срабатывание других триггеров. Этот мощный инструмент требует тонкого и согласованного применения, чтобы не получился бесконечный цикл срабатывания триггеров.

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

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

Для написания хранимых процедур и триггеров используется расширение стандартного языка SQL, так называемый встроенный SQL.

Недостатком данной модели является очень большая загрузка сервера. Действительно, сервер обслуживает множество клиентов и выполняет следующие функции:

– осуществляет мониторинг событий, связанных с описанными триггерами;

– 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 все функции СУБД: доступ к данным, контроль и поддержку целостности данных в БД, контроль доступа, обеспечение корректной параллельной работы всех пользователей с единой БД.

Если на сервер переложена большая часть бизнес-логики приложений, то требования к клиентам в этой модели резко уменьшаются. Иногда такую модель называют моделью с «тонким клиентом», в отличие от предыдущих моделей, где на клиента возлагались гораздо более серьезные задачи. Эти модели называются моделями с «толстым клиентом».

1.3. Модели серверов баз данных

При распределенной обработке данных большое значение имеет организация взаимодействия процессов типа «клиент» и процессов типа «сервер». От эффективности его реализации зависит эффективность работы системы в целом.

Организация взаимодействия клиента и сервера определяется структурой реализации серверных процессов, которая часто называется архитектурой сервера баз данных.

Первоначально существовала модель, когда управление данными (функция сервера) и взаимодействие с пользователем

были совмещены в одной программе. Это можно назвать нулевым этапом развития серверов БД.

Затем функции управления данными были выделены в самостоятельную группу - сервер, однако модель взаимодействия пользователя с сервером соответствовала парадигме «один-к-одному» (рис. 1.7), то есть сервер обслуживал запросы только одного пользователя (клиента), и для обслуживания нескольких клиентов нужно было запустить эквивалентное число серверов.

Выделение сервера в отдельную программу было революционным шагом, который позволил, в частности, поместить сервер на одну машину, а программный интерфейс с пользователем — на другую, осуществляя взаимодействие между ними по сети. Однако необходимость запуска большого числа серверов для обслуживания множества пользователей сильно ограничивала возможности такой системы.

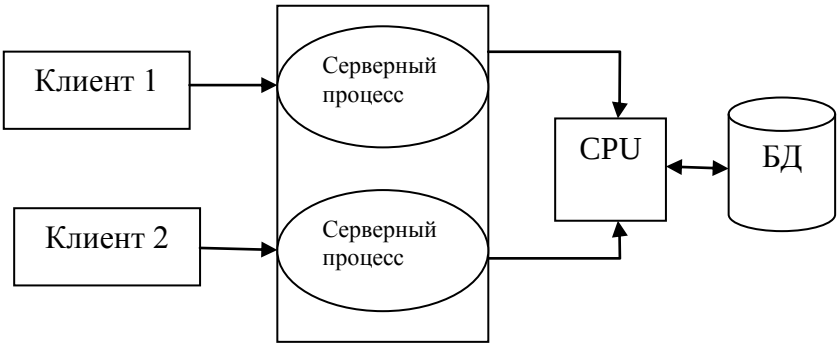

Рис. 1.7. Взаимодействие пользовательских и клиентских процессов в модели «один-к-одному»

Для обслуживания большого числа клиентов на сервере должно быть запушено большое количество одновременно работающих серверных процессов, а это резко повышало требования к ресурсам ЭВМ, на которой запускались все серверные процессы. Кроме того, каждый серверный процесс в этой моде-

ли запускался как независимый, поэтому если один клиент сформировал запрос, который был только что выполнен другим серверным процессом для другого клиента, то запрос, тем не менее, выполнялся повторно. В такой модели весьма сложно обеспечить взаимодействие серверных процессов. Эта модель самая простая, и исторически она появилась первой.

Проблемы, возникающие в модели «один-к-одному», решаются в архитектуре «систем с выделенным сервером», который способен обрабатывать запросы от многих клиентов. Сервер единственный обладает монополией на управление данными и взаимодействует одновременно со многими клиентами (рис. 1.8). Логически каждый клиент связан с сервером отдельной нитью («thread»), или потоком, по которому пересылаются запросы. Такая архитектура получила название многопотоковой односерверной («multi-threaded»).

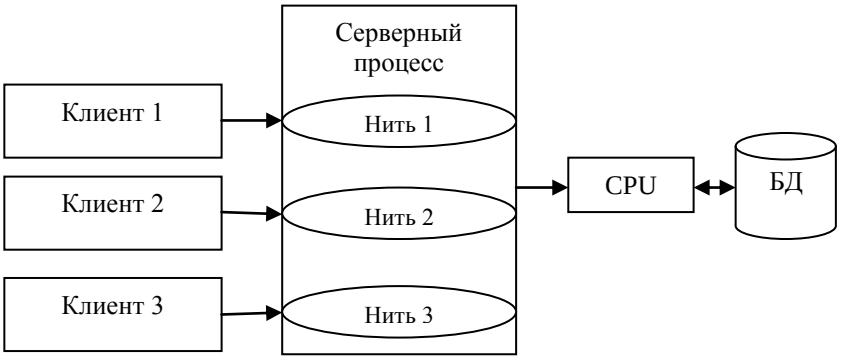

Рис. 1.8. Многопотоковая односерверная архитектура

Она позволяет значительно уменьшить нагрузку на операционную систему, возникающую при работе большого числа пользователей («trashing»).

Кроме того, возможность взаимодействия с одним сервером многих клиентов позволяет в полной мере использовать

разделяемые объекты (начиная с открытых файлов и кончая данными из системных каталогов), что значительно уменьшает потребности в памяти и общее число процессов операционной системы. Например, системой с архитектурой «один-к-одному» будет создано 100 копий процессов СУБД для 100 пользователей, тогда как 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 используют только один из них, не загружая оставшиеся три.

В некоторых системах эта проблема решается вводом промежуточного диспетчера. Подобная архитектура называется архитектурой виртуального сервера («virtual server») (рис. 1.9).

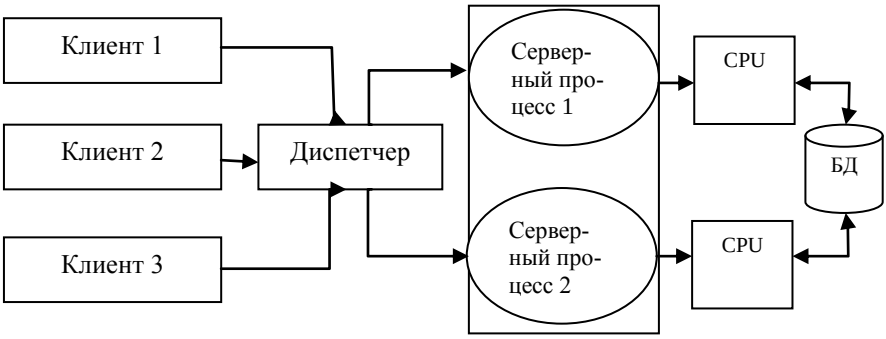

Рис. 1.9. Архитектура с виртуальным сервером

В этой архитектуре клиенты подключаются не к реальному серверу, а к промежуточному звену, называемому диспетчером, который выполняет только функции диспетчеризации запросов к актуальным серверам. В этом случае нет ограничений на использование многопроцессорных платформ. Количество

актуальных серверов может быть согласовано с количеством процессоров в системе.

Однако и эта архитектура имеет недостатки. В систему добавляется новый слой, который размещается между клиентом и сервером, что увеличивает затраты ресурсов на поддержку баланса загрузки актуальных серверов («load balancing») и ограничивает возможности управления взаимодействием «клиент—сервер». Во-первых, становится невозможным направить запрос от конкретного клиента конкретному серверу, во-вторых, серверы становятся равноправными – нет возможности устанавливать приоритеты для обслуживания запросов.

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, то это многопотоковая архитектура с несколькими серверами, представленная на рис. 1.10.

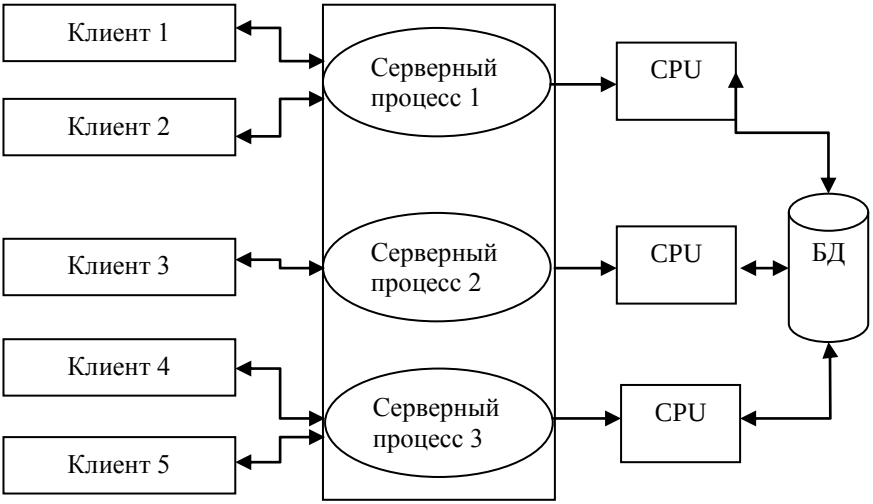

Рис. 1.10. Многопотоковая мультисерверная архитектура

Она также может быть названа многонитевой мультисерверной архитектурой. Эта архитектура связана с вопросами распараллеливания выполнения одного пользовательского запроса несколькими серверными процессами.

Существует несколько возможностей распараллеливания выполнения запроса. В этом случае пользовательский запрос разбивается на ряд подзапросов, которые могут выполняться параллельно, а результаты их выполнения потом объединяются в общий результат выполнения запроса. Тогда для обеспечения оперативности выполнения запросов их подзапросы могут быть направлены отдельным серверным процессам, а потом полученные результаты объединены в общий результат. В данном случае серверные процессы не являются независимыми процессами, такими, как рассматривались ранее. Эти серверные процессы принято называть нитями (treads), и управление нитями множества запросов пользователей требует дополнительных расходов от СУБД, однако при оперативной обработке информации в хранилищах данных такой подход наиболее перспективен.

Типы параллелизма. Существует несколько путей распараллеливания запросов.

Горизонтальный параллелизм. Этот параллелизм возникает тогда, когда хранимая в БД информация распределяется по нескольким физическим устройствам хранения — нескольким дискам. При этом информация из одного отношения разбивается на части по горизонтали (рис. 1.11). Этот вид параллелизма иногда называют распараллеливанием или сегментацией данных. И параллельность здесь достигается путем выполнения одинаковых операций, например фильтрации, над разными физическими хранимыми данными. Эти операции могут выполняться параллельно разными процессами, они независимы. Результат выполнения целого запроса складывается из результатов выполнения отдельных операций.

Время выполнения такого запроса при соответствующем сегментировании данных существенно меньше, чем время вы-

полнения этого же запроса традиционными способами одним процессом.

Вертикальный параллелизм. Этот параллелизм достигается конвейерным выполнением операций, составляющих запрос пользователя. Этот подход требует серьезного усложнения в модели выполнения реляционных операций ядром СУБД. Он предполагает, что ядро СУБД может произвести декомпозицию запроса, базируясь на его функциональных компонентах, и при этом ряд подзапросов может выполняться параллельно, с минимальной связью между отдельными шагами выполнения запроса.

И третий вид параллелизма является гибридом двух ранее рассмотренных.

Наиболее активно применяются все виды параллелизма в OLAP-приложениях, где эти методы позволяют существенно сократить время выполнения сложных запросов над очень большими объемами данных.

1.4. Трехзвенные модели организации данных

Трехзвенная модель распределения функций представляет собой типовой вариант, при котором каждая из трех функций (управление данными, обработка, представление) реализуются на отдельных компьютерах.

Данная модель имеет название – **модель сервера приложений**, или **AS-модель** (Application Server) и показана на рис. 1.11.

Согласно трехзвенной AS-модели процесс, отвечающий за организацию диалога с конечным пользователем, реализует функции представления информации и взаимодействует с компонентом приложения.

Компонент приложения, располагаясь на отдельном компьютере, в свою очередь, связан с компонентом управления данными.

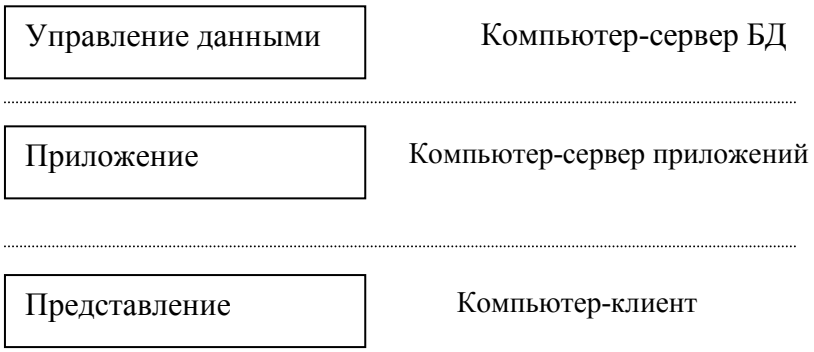

Рис. 1.11. Трехзвенная модель сервера приложений

Центральным звеном AS-модели является сервер приложений. На сервере приложений реализуется несколько прикладных функций, каждая из которых оформлена как служба предоставления услуг всем требующим этого программам. Серверов приложений может быть несколько, причем каждый из них предоставляет свой вид сервиса. Любая программа, запрашивающая услугу у сервера приложений, является для него клиентом. Поступающие от клиентов к серверам запросы помещаются в очередь, из которой выбираются в соответствии с некоторыми правилами, например, по приоритетам.

Компонент, реализующий функции представления и являющийся клиентом для сервера приложений, в этой модели трактуется более широко. Он может служить для организации интерфейса с конечным пользователем, обеспечивать прием данных от устройств, например, датчиков, или быть произвольной программой.

Достоинством AS-модели является гибкость и универсальность вследствие разделения функций приложения на три независимые составляющие. Во многих случаях эта модель оказывается более эффективной по сравнению с двухзвенными. Основной недостаток модели – более высокие затраты ресурсов

компьютеров на обмен информацией между компонентами приложения по сравнению с двухзвенными моделями.

Модель монитора транзакций. AS-модель описывает взаимодействие трех основных элементов: клиента, сервера приложений и сервера БД, но не затрагивает вопросы организации функционирования программного обеспечения при обработке информации, в частности при выполнения транзакций. Для преодоления этого недостатка предложена модель монитора транзакций.

Мониторы обработки транзакций (Transaction Processing Monitor – TPM), или мониторы транзакций, представляют собой программные системы категории промежуточного слоя (Middleware), обеспечивающие эффективное управление информационно-вычислительными ресурсами в распределенной вычислительной системе. Основное их назначение – организация гибкой, открытой среды для разработки и управления мобильными приложениями, оперативно обрабатывающими распределенные транзакции. Применение монитора транзакций наиболее эффективно в гетерогенных вычислительных системах.

Принципы организации обработки информации с помощью монитора транзакций описываются моделью монитора транзакций X / Open DTP (Distributed Transaction Processing – обработка распределенных транзакций). Эта модель включает в себя три объекта: прикладную программу, менеджер ресурсов (Resource Manager – RM) и монитор, или менеджер транзакций (Transaction Manager – TM).

В качестве прикладной программы может выступать произвольная программа-клиент.

Менеджер ресурсов RM выполняет функции сервера ресурса некоторого вида. Прикладная программа взаимодействует с RM с помощью набора специальных функций либо посредством операторов SQL (когда сервером является сервер БД). Функции менеджера ресурсов обычно выполняют серверы БД.

В задачах организации управления обработкой распределенных транзакций (транзакций, затрагивающих программные объекты вычислительной сети) ТМ взаимодействует с RM, который должен поддерживать протокол двухфазной фиксации транзакций и удовлетворять стандарту X / Open XA. Примерами СУБД, поддерживающих протокол двухфазной фиксации транзакций, являются Oracle 7.0, Open INGRES, Informix-Online 5.0.

Понятие транзакции в TPM (монитор транзакций) несколько шире, чем в СУБД, где транзакция включает в себя элементарные действия над данными базы. Здесь транзакция может охватывать и другие необходимые действия: передачу сообщения, запись информации в файл, опрос датчиков и т.д. Транзакции, которые поддерживают TPM, называют также прикладными или бизнес-транзакциями.

Для **разработчиков приложений** мониторы обработки транзакций TPM предоставляют удобства, связанные с возможностью декомпозиции приложений по нескольким функциональным уровням со стандартными интерфейсами, что позволяет создавать легко модифицируемые ИС со стройной архитектурой.

Для **пользователей распределенных систем** TPM позволяют улучшить показатели пропускной способности и времени отклика, снизить стоимость обработки данных в оперативном режиме на основе эффективной организации вычислительного процесса.

Администраторы распределенных систем, имея TPM, получают возможность легкого масштабирования ИС и увеличения производительности обработки информации. Без остановки серверов приложений в любое время может быть добавлен, например, компьютер или менеджер ресурсов.

1.5. Распределенные базы данных

Распределенная база данных (РБД) – система логически интегрированных и территориально распределенных БД, язы-

ковых, программных, технических и организационных средств, предназначенных для создания, ведения и обработки информации [2].

Распределенная база данных предполагает хранение данных на нескольких узлах сети, обработку данных и их передачу между узлами сети при выполнении запросов к данным. В распределенной базе разбиение данных может осуществляться разными способами:

- хранение различных таблиц на разных компьютерах;
- хранение разных фрагментов одной таблицы на разных компьютерах.

Для пользователя или приложения не должно иметь значение, каким образом данные распределены между компьютерами. Работа с распределенной базой данных должна осуществляться также, как и с централизованной.

К достоинствам распределенных баз данных (РБД) можно отнести следующее:

- соответствие структуры РБД структуре организации;
- гибкое взаимодействие локальных БД;
- непосредственный доступ к информации, снижение стоимости передачи данных;
- малое время отклика за счет распараллеливания процессов, высокая надежность;
- возможность распределения информации в соответствии с активностью ее использования;
- независимая параллельная разработка локальных БД.

Вместе с тем РБД обладают более сложной структурой, что вызывает появление дополнительных проблем:

- избыточность данных;
- несогласованность данных во времени;
- необходимость согласования процессов обновления и запросов;
- использование телекоммуникационных ресурсов;
- стандартизация общего интерфейса;
- реализация взаимодействия локальных БД;

- усложнение защиты данных.

Локальные базы данных, объединяемые в распределенную базу данных, могут быть:

- гомогенными (однородными), чаще всего реляционными, созданными с использованием СУБД одного производителя;

- гетерогенными (неоднородными), построенными с использованием разных моделей данных и созданных с использованием СУБД разных производителей.

Крупнейший специалист в области разработки теории баз данных Дейт К. определил две основные идеи распределенных информационных систем:

- работа множества пользователей с общей БД;

- объединение распределенных данных на логическом и физическом уровнях в общую БД.

Существуют следующие основные принципы создания и функционирования распределенных БД:

- прозрачность размещения данных для пользователя; распределенная БД должна представляться пользователю как нераспределенная;

- изолированность пользователей друг от друга; на работу одного пользователя с БД не должна влиять работа с БД другого пользователя;

- синхронизация БД и непротиворечивость хранимых данных в любой момент времени.

Дейт К. сформулировал 12 правил организации распределенных баз данных [4].

Локальная автономия. Означает, что управление данными в каждом узле распределенной системы выполняется локально. С одной стороны, база данных, расположенная на одном из узлов, является компонентом распределенной системы, фрагментом общего пространства данных. С другой стороны, база данных функционирует как полноценная локальная база данных, управление ею осуществляется локально и независимо от других узлов системы.

Независимость узлов. Все узлы равноправны и независимы, а расположенные на них БД являются равноправными поставщиками данных в общее пространство данных. База данных на каждом из узлов самодостаточна, включает полный собственный словарь данных и полностью защищена от несанкционированного доступа.

Непрерывность операций. Означает возможность непрерывного доступа к данным в рамках распределенной базы данных независимо от их расположения и операций обработки, выполняемых на локальных узлах.

Прозрачность расположения. Пользователь, обращающийся к базе данных, ничего не должен знать о реальном, физическом размещении данных в узлах информационной системы.

Прозрачная фрагментация. Возможность распределенного на различных узлах размещения данных, логически представляющих собой единое целое. Существует фрагментация двух типов: горизонтальная и вертикальная. Первая означает, что строки таблицы хранятся на различных узлах. Вторая означает распределение столбцов таблицы по нескольким узлам.

Прозрачное тиражирование. Тиражирование данных – это асинхронный процесс переноса изменений объектов исходной базы данных в базы, расположенные на разных узлах распределенной системы.

Обработка распределенных запросов. Возможность выполнения операций выборки данных из распределенной базы данных с помощью запросов, сформулированных на языке SQL.

Обработка распределенных транзакций. Возможность выполнения операций обновления распределенной базы данных, не нарушающих целостность и согласованность данных. Эта цель достигается применением двухфазного протокола фиксации транзакций.

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

Независимость от операционных систем. Это качество вытекает из предыдущего и означает многообразие операционных систем, управляющих узлами распределенной системы.

Прозрачность сети. Доступ к любым базам данных осуществляется по сети. Спектр поддерживаемых конкретной СУБД сетевых протоколов не должен быть ограничением системы, основанной на распределенной БД.

Независимость от СУБД. Это качество означает, что в распределенной системе могут работать СУБД различных производителей, и возможны операции поиска и обновления в базах данных различных моделей и форматов.

Существует терминология, которая связана с распределенными базами данных [2].

Пользователь БД – программа или человек, обращающийся к БД на языке манипулирования данными (ЯМД).

Запрос – процесс обращения пользователя к БД с целью ввода, получения или изменения информации в БД.

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

Логическая структура БД – определение БД на физически независимом уровне, ближе всего соответствует концептуальной модели БД.

Топология БД = Структура распределенной БД – схема распределения физической БД по сети.

Локальная автономность – означает, что информация локальной БД и связанные с ней определения данных принадлежат локальному владельцу и им управляются.

Удаленный запрос – запрос, который выполняется с использованием модемной связи.

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

Поддержка распределенной транзакции – допускает обработку транзакции, состоящей из нескольких запросов SQL,

которые выполняются на нескольких узлах сети (удаленных или локальных), но каждый запрос в этом случае обрабатывается только на одном узле, то есть запросы не являются распределенными. При обработке одной распределенной транзакции разные локальные запросы могут обрабатываться в разных узлах сети.

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

1.6. Управление распределенными данными

С управлением данными в распределенных системах связаны следующие две группы проблем: поддержка соответствия БД вносимым изменениям и обеспечение совместного доступа нескольких пользователей к общим данным.

Поддержка соответствия БД вносимым изменениям. В современных распределенных системах информация может храниться централизовано или децентрализовано. В первом случае проблемы идентичности представления информации для всех пользователей не существует, так как все последние изменения хранятся в одном месте. На практике информация может изменяться одновременно в нескольких узлах распределенной вычислительной системы. В этом случае возникает проблема контроля за всеми изменениями информации и предоставления ее в достоверном виде всем пользователям.

Существует две основные технологии децентрализованного управления БД: распределенных БД (Distributed Database) и тиражирования, или репликации, БД (Data Replication).

Распределенная БД состоит из нескольких фрагментов, размещенных на разных узлах сети и, возможно, управляемых разными СУБД. С точки зрения программ и пользователей, обращающихся к распределенной БД, последняя воспринимается как единая локальная БД (рис. 1.12).

Информация о местоположении каждой из частей распределенной БД и другая служебная информация хранится в так

называемом глобальном словаре данных. В общем случае этот словарь может храниться на одном из узлов или тоже быть распределенным.

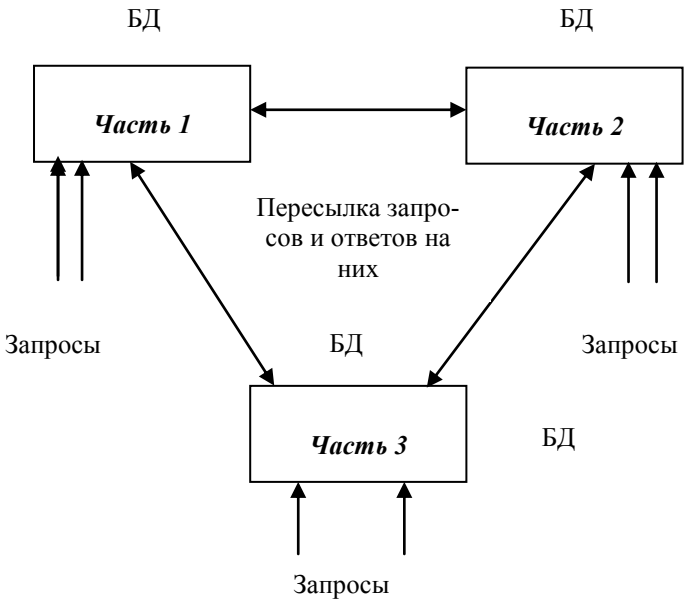

Рис. 1.12. Модель распределенной БД

Для обеспечения корректного доступа к распределенной БД в современных системах чаще всего применяется протокол (метод) двухфазной фиксации транзакций (two-phase commit). Сущность этого метода состоит в двухэтапной синхронизации выполняемых изменений на всех задействованных узлах. На первом этапе в узлах сети производятся изменения в их БД, о чем посылаются уведомления компоненту системы, управляющему обработкой распределенных транзакций.

На втором этапе, получив от всех узлов сообщения о правильности выполнения операций (что свидетельствует об от-

сутствии сбоев и отказов аппаратно-программного обеспечения), управляющий компонент выдает всем узлам команду фиксации изменений. После этого транзакция считается завершенной, ее результат необратимым.

Основным достоинством модели распределенной БД является то, что пользователи всех узлов (при исправных коммуникационных средствах) получают информацию с учетом всех последних изменений. Второе преимущество состоит в экономном использовании внешней памяти компьютеров, что позволяет организовать БД больших объемов.

К недостаткам модели распределенной БД относится следующее: жесткие требования к производительности и надежности каналов связи, а также большие затраты коммуникационных и вычислительных ресурсов из-за их связывания на все время выполнения транзакций. При интенсивных обращениях к распределенной БД, большом числе взаимодействующих узлов, низкоскоростных и ненадежных каналах связи обработка запросов по этой схеме становится практически невозможной.

Модель тиражирования данных, в отличие от технологии распределенных БД, предполагает дублирование данных (создание точных копий) в узлах сети (рис. 1.13).

Данные всегда обрабатываются как обычные локальные. Поддержку идентичности копий друг другу в асинхронном режиме обеспечивает компонент системы, называемый репликатором (replicator). При этом между узлами сети могут передаваться как отдельные изменения, так и группы изменений. В течении некоторого времени копии БД могут отличаться друг от друга.

К основным достоинствам модели тиражирования БД (в сравнении с предыдущей моделью) относятся: более высокая скорость доступа к данным, так как они всегда есть в узле; существенное уменьшение передаваемого по каналам связи потока информации, поскольку происходит передача не всех операций доступа к данным, а только изменений в БД; повышение надежности механизмов доступа к распределенным данным,

поскольку нарушение связи не приводит к потере работоспособности системы.

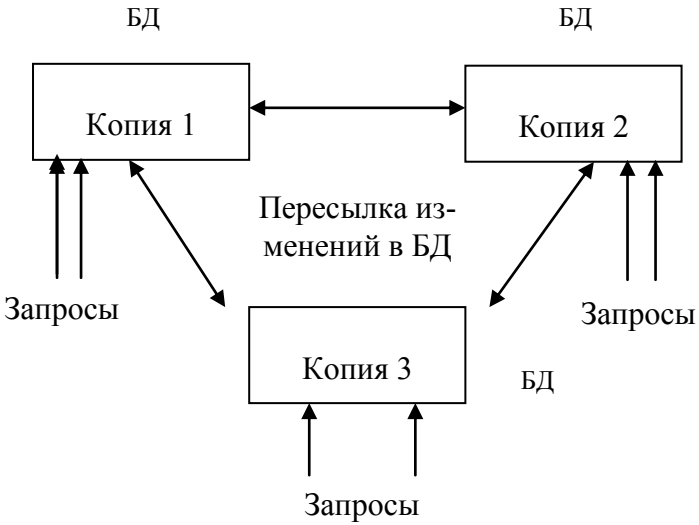

Рис. 1.13. Модель тиражирования БД

Основной недостаток модели тиражирования БД заключается в том, что на некотором интервале времени возможно «расхождение» копий БД. Если отмеченный недостаток некритичен для прикладных задач, то предпочтительно иметь схему с тиражированием БД.

Доступ к общим данным. При обслуживании обращений к общим данным средства управления БД должны обеспечивать по крайней мере два основных метода доступа: монопольный и коллективный.

Основными объектами доступа в различных системах могут быть целиком БД, отдельные таблицы, записи, поля записей, элементы приложения такие, как отчеты, запросы, экранные формы.

Монопольный доступ обычно используется в двух случаях:

во-первых, когда требуется исключить доступ к объектам со стороны других пользователей (например, при работе с конфиденциальной информацией);

во-вторых, когда производятся ответственные операции с БД, не допускающие других действий, например, изменение структуры БД.

В первом случае пользователь с помощью диалоговых средств СУБД или прикладной программы устанавливает явную блокировку. Во втором случае пользователь также может установить явную блокировку, либо положиться на СУБД. Последняя обычно автоматически устанавливает неявную (без ведома пользователя или приложения) блокировку, если это необходимо.

В режиме **коллективного доступа** полная блокировка на используемые объекты, как правило, не устанавливается. Коллективный доступ возможен, например, при одновременном просмотре таблиц. Попытки получить монопольный доступ к объектам коллективного доступа должны быть пресечены. Например, в ситуации когда один или несколько пользователей просматривают таблицу, а другой пользователь собирается удалить эту же таблицу.

Для организации коллективного доступа в СУБД применяется **механизм блокировок**. Суть блокировки состоит в том, что на время выполнения какой-либо операции в БД доступ к используемому объекту со стороны других потребителей временно запрещается или ограничивается. Например, при копировании таблицы она блокируется от изменения, хотя и разрешено просматривать ее содержимое.

Рассмотрим типичный набор блокировок. В конкретных программах схемы блокирования объектов могут отличаться от описываемых.

Выделим четыре вида блокировок, перечисленных в порядке убывания строгости ограничений на возможные действия:

полная блокировка;

блокировка от записи;

предохраняющая блокировка от записи;

предохраняющая полная блокировка.

Полная блокировка. Означает полное завершение всяких операций над основными объектами (таблицами, отчетами и экранными формами). Этот вид блокировок обычно применяется при изменении структуры таблицы.

Блокировка от записи. Накладывается в случаях, когда можно использовать таблицу, но без изменения ее структуры или содержимого. Такая блокировка применяется, например, при выполнении операции слияния данных из двух таблиц.

Предохраняющая блокировка от записи. Предохраняет объект от наложения на него со стороны других операций полной блокировки, либо блокировки от записи. Этот вид блокировки позволяет тому, кто раньше «захватил» объект, успешно завершить модификацию объекта. Предохраняющая блокировка о записи совместима с аналогичной блокировкой (предохраняющей блокировкой от записи), а также с предохраняющей полной блокировкой. Примером необходимости использования этой блокировки является режим совместного редактирования таблицы несколькими пользователями.

Предохраняющая полная блокировка. Предохраняет объект от наложения на него со стороны других операций только полной блокировки. Обеспечивает максимальный уровень совместного использования объектов. Такая блокировка может использоваться, например, для обеспечения одновременного просмотра несколькими пользователями одной таблицы. В группе пользователей, работающих с одной таблицей, эта блокировка не позволит никому изменить структуру общей таблицы.

При незавершенной операции с некоторым объектом и запросе на выполнение новой операции с этим же объектом производится попытка эти операции совместить. Совмещение возможно тогда, когда совместимыми оказываются блокировки, накладываемые конкурирующими операциями.

В отношении перечисленных выше четырех блокировок действуют следующие правила совмещения:

при наличии полной блокировки над объектом нельзя производить операции, приводящие хотя бы к одному из видов блокировок (полная блокировка несовместима ни с какой другой блокировкой);

блокировка от записи совместима с аналогичной блокировкой и предохраняющей полной блокировкой;

предохраняющая блокировка от записи совместима с обоими видами предохраняющих блокировок;

предохраняющая полная блокировка совместима со всеми блокировками, кроме полной.

Обычно в СУБД каждой из выполняемых с БД операций соответствует определенный вид блокировок, которую эта операция накладывает на объект. Пользователям современных СУБД , работающим в интерактивном режиме, не нужно помнить все тонкости механизма блокировки, поскольку система достаточно «разумно» осуществляет автоматическое блокирование во всех случаях, когда это требуется. При этом система сама стремится предоставить всем пользователям наиболее свободный доступ к объектам. При необходимости можно воспользоваться командными и языковыми средствами явного определения блокировок.

Тупики. Если не управлять доступом к совместно используемым объектам, то между потребителями ресурсов могут возникать тупиковые ситуации. Следует отличать понятие блокировки в смысле контроля доступа к объектам от блокировки в смысле тупикового события.

Существует два основных вида тупиков: взаимные (deadlock) и односторонние (livelock).

Простейшим случаем **взаимного тупика** является ситуация, когда каждый из двух пользователей стремится захватить данные, уже захваченные другим пользователем (рис. 1.14). В этой ситуации пользователь-1 ждет освобождения ресурса N, в то время как пользователь-2 ожидает освобождения от захвата ресурса M. Следовательно, никто из них не может продолжить работу.

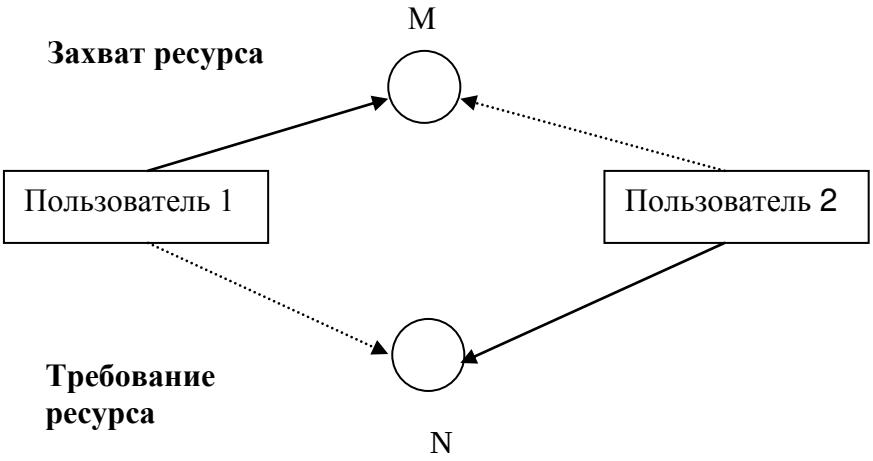

Рис. 1.14. Пример взаимного тупика

Возможны и более сложные ситуации, когда выполняются обращения трех и более пользователей к нескольким ресурсам.

Односторонний тупик возникает в случае требования получить монопольный доступ к некоторому ресурсу как только он станет доступным и невозможности удовлетворить это требование.

Системы управления распределенными БД, очевидно должны иметь соответствующие средства обнаружения или предотвращения конфликтов, а также разрешения возникающих конфликтов.

1.7. Разработка распределенных баз данных

Распределенная база данных состоит из связанных локальных баз данных (ЛБД). Возможно два варианта создания РБД: проектирование с самого начала; объединение (интеграция) уже готовых ЛБД. Модели данных в ЛБД могут быть одинаковы (однородные ЛБД), или различны (неоднородные ЛБД).

В качестве принципов проектирования РБД можно использовать следующие:

- максимум локализации данных и сокращение количества пересылаемых данных;
- локально расположенные данные должны работать с наибольшим числом приложений.

В качестве критериев проектирования РБД могут быть выбраны следующие:

- минимум объема пересылаемых данных;
- минимум стоимости трафика;
- минимум времени на обслуживание запросов к базе данных.

При проектировании РБД учитывается возможность работы с одним или несколькими приложениями.

Возможно использование восходящего и нисходящего проектирования.

Восходящее проектирование обычно используется в случае, когда РБД создается из уже работающих локальных БД.

При проектировании РБД возможны проблемы:

- ошибки в создании структуры локальных БД и их заполнении;
- просчеты в организации процедур фрагментации и локализации;
- системные ошибки в программном обеспечении взаимодействия локальных БД при реализации одновременного доступа;
- необходимость восстановления РБД в случае неисправности технических средств.

Этапы проектирования РБД совпадают с этапами проектирования централизованной БД. Отличие наблюдается в этапах фрагментации (разделения базы данных на ЛБД) и локализации (размещения ЛБД в узлах сети).

На фрагментацию базы данных влияют следующие факторы:

- допустимый размер каждого раздела;
- модели данных;
- частота использования приложений;
- структурная совместимость;
- производительность БД.

Реализация размещения БД в узлах сети является многовариантной задачей. На практике рекомендуется упростить решаемую задачу, например: допустить временное рассогласование фрагментов БД в узлах, осуществление процедуры обновления БД из одного узла.

Фрагментация может быть горизонтальной и вертикальной. Фрагмент может быть результатом реализации операций селекции и/или проекции реляционной алгебры.

При декомпозиции базы данных следует выполнить ряд условий.

Полнота. Все данные глобального отношения должны быть отображены в его фрагментах.

Восстанавливаемость. Возможность восстановить глобальное отношение из фрагментов.

Непересечение. Целесообразно, чтобы фрагменты базы данных не пересекались. Хотя дублирование данных возникает на этапе локализации.

При горизонтальной фрагментации, которая реализуется с помощью селекции, формируется подмножество кортежей, имеющих общие свойства.

При вертикальной фрагментации, которая реализуется с помощью проекции, глобальное отношение делится на более мелкие отношения для локальных приложений.

Фрагментация корректна, если любой атрибут глобального отношения присутствует в каком-либо подмножестве атрибутов новых отношений. Глобальное отношение восстанавливается естественным соединением из новых отношений. Иными словами декомпозиция глобального отношения должна обладать свойством соединения без потерь.

Возможна смешанная фрагментация, которая предполагает одновременное использование селекции и проекции.

После фрагментации осуществляют локализацию (размещение). Задача размещения фрагментов базы данных в узлах сети является оптимизационной задачей. В качестве критериев локализации используют частоту использования фрагментов базы данных приложениями в узлах сети (максимум числа ссылок), или минимальный объем передаваемых данных, или минимальную общую стоимость трафика.

1.8. Использование и функционирование РБД

Особенности использования РБД связаны со спецификой реализации запросов. Функционирование РБД связано с реализацией одновременного доступа, защиты, восстановления данных.

Запросы. Запросы могут быть заранее известными и неизвестными.

Если структура запросов заранее известна, то эффективность реализации запросов строится на применении фрагментации и локализации.

Если же структура запросов неизвестна, возникает необходимость оптимизации процессов реализации запросов. В качестве критериев для оптимизации процессов реализации запросов могут быть выбраны следующие:

- минимум времени передачи данных;
- минимум времени обработки данных в узлах;
- увеличение параллельности при обработке и передачах данных;

- улучшение трафика и загрузки процессоров в сети;

- минимум стоимости (задержки) передачи данных (если имеются низкоскоростные каналы связи);

- минимум стоимости локальной обработки (если скорость передачи данных соизмерима с быстродействием процессора);

- выравнивание нагрузки компьютеров.

Стоимость обработки данных зависит от стоимости установки связи и от стоимости передачи единицы информации и объема передаваемых данных.

Задержка данных зависит от времени установления связи и времени передачи единицы информации и объема передаваемых данных.

Стоимость и время пересылки данных можно сокращать за счет снижения объема передаваемой информации.

Часто алгоритм оптимизации закладывается в систему управления РБД в виде приложений или утилит.

Одновременный доступ. Особенностью РБД является многопользовательский распределенный доступ к данным.

Взаимодействие элементов в РБД осуществляется посредством распределенных транзакций. В транзакции выделяют две команды: читать и писать.

Для распределенных транзакций справедливы те же понятия, что и в централизованных БД: синхронизация, расписание, конфликт, рестарт. Две транзакции вступают в конфликт, если они работают с одними и теми же данными и одна из них реализует операцию «писать».

В системах управления РБД (СУРБД) в каждом узле ведется свое расписание. Вектор расписаний всех узлов называют распределенным расписанием.

Существует несколько методов синхронизации производимых изменений в узлах распределенной базы данных [2]:

- блокировка;

- голосование по большинству;

- метод предварительного анализа конфликтов.

Блокировка. Блокировка может осуществляться в главном узле, возможна блокировка с помощью предикатов или блокировка с главной копией.

Выделенный узел становится приоритетным и управляет остальными узлами как в центральной БД. Блокировка в централизованной БД предусматривается для режима записи, а чтение предполагает коллективный режим работы. Эти режимы используются и в блокировке с главным узлом, который руководит процессом синхронизации.

Применяют двух- и трехфазную транзакции. В двухфазной транзакции выделяют:

- блокировку ресурса;
- разблокировка (возвращение ресурса и снятие блокировки).

Только главный узел выдает запрос на блокировку, который проходит по всей сети.

Действия по изменению данных допускаются только над предварительно заблокированными данными, при этом сначала должно быть проведено обновление всех копий. В таком режиме возможны тупики, которые могут устраняться снятием с обслуживания позже поступивших транзакций.

Двухфазная блокировка работает следующим образом.

В фазе блокировки ресурса транзакция должна затребовать предикат-блокировку. Узел-координатор ведет таблицу блокировок. Блокировка осуществляется, если она не вступает в конфликт с другими блокировками. В противном случае транзакция переводится в режим ожидания или ей разрешается осуществить перехват ресурса.

В фазе разблокировки осуществляется удаление строки из таблицы блокировок.

Тупиковые ситуации могут быть устранены в трехфазной транзакции (блокировка, выполнение, разблокировка).

Блокировка с главным узлом и с использованием предикатов сводятся к централизации процедуры синхронизации. В

этом случае производительность БД ограничена возможностями центрального узла.

Дальнейшее развитие синхронизации пошло в направлении децентрализации. При децентрализованном управлении имеется избыточность информации и ни один узел не является главным. Транзакция, если это необходимо, блокирует копии данных в каждом узле.

Голосование по большинству. Существует избыточность РБД: в любом узле есть копия любого элемента.

Транзакция выполняется лишь в одном узле:

- команда чтения в своем узле работает без блокировок и синхронизации;

- при выполнении команды записи имя элемента и его новое значение заносятся в список обновлений.

После завершения транзакции список попадает в каждый узел и узлы голосуют за выполнение списка обновлений. В результате обновление реализуется, либо попадает в очередь на ожидание выполнения. При этом обновление может попасть в разное место очереди ожидания.

Если большинство «за», транзакция принимается и обновление проводится во всех узлах.

Узел голосует «за», если:

1) элемент данных, прочитанный транзакцией, не был изменен после чтения;

2) транзакция не конфликтует с другой транзакцией, которая ожидает решения, если данный узел проголосовал «за», но другая транзакция еще не была принята или отвергнута в целом в данном узле.

Для определения очередности выполнения транзакций используются временные метки (метки времени). Метка присваивается транзакции и каждому элементу данных (последней транзакции, которая его обновила).

Когда узел принимает список обновления, он сравнивает временные метки. Если условие 1 не выполняется, то узел отвергает транзакцию, которая рестартует. Если условие 1 выпол-

няется, а 2 – нет, то узел не может голосовать «за» эту транзакцию, которая ожидает решения, пока его не получит. При выполнении обоих условий узел голосует «за».

Поскольку разные узлы получают списки обновлений в разном порядке, они и голосуют в разном порядке. Это может приводить к тупикам. Чтобы их избежать, узел голосует «против», если условие 1 выполняется, а 2 – нет и транзакция получает большую временную метку, чем ожидающая транзакция.

Метод предварительного анализа конфликтов. До сих пор синхронизировались все конфликтующие операции записи и чтения. Однако не все конфликты требуют синхронизации.

Все транзакции делятся на классы. С любым классом связан модуль транзакций (МТ), который является частью СУБД. В этом случае могут конфликтовать только транзакции разных классов и их необходимо синхронизировать. Конфликт классов моделируется графом конфликтов, вершины которого соответствуют множествам чтения и записи класса, а ребра представляют собой конфликты.

Граф конфликтов создается и анализируется статически, когда определены БД и классы. СУБД определяет, какие классы должны синхронизировать свои транзакции.

1.9. Защита данных, восстановление РБД

Защита. Для защиты данных администратор БД определяет уровни доступа пользователей к данным в РБД и права на выполнение конкретных операций над данными. При этом пользователь получает идентификатор и пароль.

Отдельные части РБД могут иметь свою защиту. Для этого администратор определяет полномочия: перечень операций, которые пользователь может выполнять с соответствующими элементами данных.

Полномочия могут относиться к объектам или системе в целом. Полномочия для отдельных объектов предоставляют для какой-либо таблицы конкретным пользователем возможность

выполнения конкретным операциям над данными (Insert, Update, Delete). Системные полномочия касаются всей БД и предполагают выполнение следующих операций: создание таблицы, изменение структуры, удаление таблицы.

В случае повышенных требований к защите данных следует использовать механизмы компьютерной безопасности, обеспечивающие конфиденциальность, целостность данных и защиту от отказов. Тремя главными механизмами защиты являются шифрование данных, контроль за процедурой загрузки и использование защитных криптографических контрольных сумм.

Восстановление. Восстановление связано с приведением системы в корректное состояние после (аппаратного) сбоя.

В РБД возможны отказы узлов или сети связи. Поскольку над помехоустойчивостью сетей работают давно и успешно, чаще всего считают сеть надежной. Поэтому считаются ненадежными узлы.

Система восстановления решает две группы задач:

- при незначительной неисправности – откат в выполнении текущей транзакции;

- при существенных отказах – минимизация работы по восстановлению РБД.

Наиболее часто для целей восстановления используется системный журнал регистрации. При применении двухфазной транзакции вводится точка фиксации, в которой принимается решение о фиксации транзакции. Для выполнения фиксации или отката используется информация журнала регистрации (блокировок, отмен и повторов).

Копирование базы данных возможно после закрытия БД. Однако для РБД с интенсивными обновлениями копирование следует проводить в процессе работы РБД. Физически РБД представляет собой группы файлов.

Процедура восстановления проводится следующим образом.

1. Устраняется аппаратный сбой.

2. Восстанавливаются испорченные файлы данных путем копирования архивных копий и групп регистрации транзакций.

3. Запускается процесс восстановления:

- с применением транзакций (применением к архивным копиям испорченных данных сведений из журнала транзакций);

- с отменой незавершенных транзакций.

Возможно выделить следующие основные состояния узла: исправный, неисправный, восстанавливаемый.

Узел в надежной сети может работать в двух вариантах: без дублирования данных и с дублированием данных.

Обычно одна транзакция делится на серию субтранзакций. При откате одной из них осуществляется откат транзакции в целом. При этом невыполненные транзакции и субтранзакции после устранения аппаратного сбоя выполняются вновь.

Откат всей транзакции не всегда нужен. Возможен другой способ: прервать только откатившуюся транзакцию, которая предпринимает фиксированное число попыток повторного выполнения. Если все они закончатся неудачей, осуществляется откат всей транзакции.

При дублировании данных (разные копии хранятся в разных узлах) в случае сбоев появляется возможность работы транзакций даже тогда, когда некоторые узлы находятся в неработающем состоянии.

Может быть частичное и полное (в каждом узле находится полная копия РБД) дублирование.

При частичном дублировании среди узлов выделяется основной узел. Транзакции нумеруются. При восстановлении вспомогательный узел должен передавать основному узлу последний зафиксированный номер транзакции. Затем он запрашивает информацию об изменениях, проведенных к данному моменту времени транзакциями с большими номерами.

Достоинствами метода основной копии являются единственная последовательность корректировки БД и уменьшение вероятности тупика.

В случае чтения во всех доступных узлах и использования активных узлов формируется и выводится список записей об активных узлах, а основной узел используется для восстановления. «Основная» транзакция при обновлении должна запрашивать обстановку для записи и корректировки данных во всех узлах списка, включая основной. При наличии отказов и восстановлений список меняется основным узлом.

2. СОЗДАНИЕ БАЗЫ ДАННЫХ СРЕДСТВАМИ MS SQL SERVER

SQL Server – это реляционная система управления базой данных, обладающая многими функциональными возможностями. Эти возможности позволяют сконфигурировать информационную систему так, чтобы она соответствовала потребностям бизнеса. Данная система может быть использована для малых предприятий, корпораций и предприятий электронного бизнеса.

Система SQL Server может быть реализована как клиент-серверная система, либо как автономная «настольная» система. Тип создаваемой системы зависит от количества пользователей, которые должны одновременно осуществлять доступ к базе данных, и от характера работ, которые должны выполняться.

Клиент-серверная система SQL Server может иметь двухзвенную либо трехзвенную установку. Независимо от варианта установки, программное обеспечение и базы данных SQL Server размещаются на центральном компьютере. Пользователи работают на отдельных компьютерах, которые называются клиентами. Доступ пользователей к базе данных производится при помощи приложений с компьютеров-клиентов (в двухзвенных системах) либо при помощи приложений, выполняющихся на специально предназначенном для этой цели компьютере, который называется сервером приложений (в трехзвенных системах).

В двухзвенных системах клиенты исполняют приложения, осуществляющие доступ к серверу базы данных непосредственно через сеть. Таким образом, компьютеры-клиенты исполняют программный код, соответствующий нуждам некоторой группы пользователей, и код, отображающий для пользователя результаты доступа к базе данных. Такие клиенты называются толстыми (thick client), потому что они выполняют два вида работы.

2.1. Структура базы данных

Каждая база данных SQL Server состоит из набора файлов. Эти файлы могут объединяться в группы файлов, что облегчает их администрирование, помогает в размещении данных и повышает производительность.

Файл базы данных может быть либо файлом данных, либо файлом журнала. Файлы данных служат для хранения данных и объектов, таких как таблицы, индексы, представления, триггеры и хранимые процедуры. Имеется два типа файлов данных: первичные и вторичные. Файлы журналов служат только для хранения информации из журналов транзакций. Место на диске, отводимое для файлов журналов, всегда должно администрироваться отдельно от места, отводимого для данных, и никогда не должно быть частью файла данных.

Первичные файлы данных содержат всю информацию для запуска базы данных и ее системных таблиц и объектов. Они указывают на другие файлы, созданные в базе данных. Каждая база данных может иметь ровно один первичный файл. Этот файл имеет расширение .mdf.

Вторичные файлы данных не являются обязательными. Они могут хранить данные и объекты, которые отсутствуют в первичном файле. База данных может вообще не иметь ни одного вторичного файла (если все ее данные хранятся в первичном файле). Можно иметь ноль, один или несколько вторичных файлов. Вторичные файлы имеют расширение .ndf. Если в про-

цессе использования базы данных планируется размещение ее на нескольких дисках, то в этом случае можно создать вторичные файлы базы данных. При нехватке свободного места для первичного файла базы данных добавляемая информация будет размещаться во вторичных файлах.

Файлы журналов транзакций хранят всю информацию из журнала транзакций, служащую для восстановления базы данных. Каждая база данных должна иметь хотя бы один файл журнала. Для этих файлов рекомендуется применять расширение .ldf.

2.2. Типы данных в MS SQL Server

Одним из основных моментов в процессе создания таблиц является определение типов данных для ее полей. Тип данных поля таблицы определяет тип информации, которая будет размещаться в этом поле. SQL-сервер поддерживает большое число различных типов данных: текстовые, числовые, двоичные и т.д.

Основные типы данных представлены в табл. 1.

Таблица 1

Основные типы данных SQL Server

Тип данных	Описание	Место в памяти
bigint	Восьмибайтное целое число (полное целое)	8 байт
char[(n)]	Символьные данные фиксированной длины (n символов, где n принимает значения от 1 до 8000)	N байт
datetime	Дата и время от 1 января 1753 года до 31 декабря 9999 года	8 байт

Продолжение табл. 1

decimal[(p,[s])] или numeric[(p,[s])]	Числа фиксированной точности и фиксированного масштаба. Точность p определяет общее количество цифр. Масштаб s определяет максимальное количество цифр справа от точки.	От 5 до 17 байт, в зависимости от точности
integer или int	Целочисленные данные (полное целое)	4 байта
money	Данные для денежных величин	8 байт
real	Числовые данные с плавающей точностью	4 байта
smallint	Целочисленные данные	2 байта
smalldatetime	Данные для даты и времени от 1 января 1900 года до 6 июня 2079 года, с точностью до одной минуты	4 байта
text	Символьные данные переменной длины не в кодировке Unicode, длиной более 8000 байт.	16 байт для указателя
tinyint	Целочисленные данные в диапазоне от 0 до 255	1 байт
varchar[(n)]	Данные переменной длины не в кодировке Unicode, длиной в n символов, где n может принимать значение от 1 до 8000	Фактическая длина введенных данных

2.3. Создание базы данных, таблиц, схемы данных средствами MS SQL Server 2005

При взаимодействии приложения с базой данных необходимо, чтобы SQL Server был запущен. Для проверки режима работы сервера запускают программу SQL Server Configuration Manager. В «Диспетчере конфигурации SQL Server» выбирают вкладку «Службы SQL Server 2005» и просматривают запущенные службы.

Для создания новой базы данных выбирают пункты: Пуск, Программы, MS SQL Server 2005, SQL Server Management Studio. В открывшемся диалоговом окне щелкают по кнопке **Соединить**, при этом происходит соединение с сервером.

В панели **Обозреватель объектов** на объекте **Базы данных** вызывают контекстное меню и выбирают пункт **Создать базу данных.**

В поле **Имя базы** данных задают имя базы данных. Во вкладке **Файлы базы данных** можно выбрать место сохранения файла. Затем щелкают по кнопке **ОК**. На дереве в окне обозревателя объектов появится имя созданной базы данных.

Щелчком по кнопке «+» раскрывают на дереве объекты базы данных. Для создания таблицы щелкают правой кнопкой мыши на объекте **Таблицы** и выбирают пункт **Создать таблицу**. Дальнейшее создание таблицы выполняют аналогично тому, как это происходит в SQL Server 2000.

Для создания схемы данных щелкают на объекте **Диаграммы** и выбирают пункт меню **Создать диаграмму БД**. В дальнейшем выбирают связываемые таблицы и перетаскивают поля связи из одной таблицы в другую.

2.4. Обеспечение доступа к базе данных средствами MS SQL Server 2005

Создание имени входа. На дереве объектов раскрывают объект **Безопасность**. На объекте **Имена входов** вызывают

контекстное меню и выбирают пункт **Создать имя входа**. В открывшемся диалоговом окне в поле **Имя входа** вводят имя, активизируют переключатель **Проверка подлинности SQL Server**, вводят пароль в окне **Пароль**, вводят подтверждение пароля; выбирают нужную базу данных по умолчанию, выбирают язык. Необходимо выключить режим **Задать срок окончания действия пароля**.

На дереве объектов выбирают **Серверные роли** и задают какую-то роль (например, sysadmin), затем щелкают по кнопке **ОК**.

Создание имени пользователя. На дереве объектов раскрывают объекты базы данных и выбирают Безопасность, а затем **Пользователи**. На объекте **Пользователи** вызывают контекстное меню и выбирают пункт **Создать пользователя**. В открывшемся диалоговом окне вводят имя пользователя и в имени входа задают созданное ранее имя входа, щелкают по кнопке **ОК**.

2.5. Перенос базы данных на другой компьютер

Исходные файлы базы данных имеют расширение .mdf и .ldf. При переносе базы данных на другой компьютер необходимо:

1) скопировать файлы с названием базы данных на новый компьютер;

2) в обозревателе объектов найти вкладку «Базы данных» и в свойствах выбрать «Присоединить»;

3) нажать кнопку «Добавить» и выбрать нужную базу данных (расширение . mdf);

4) в свойствах базы данных во вкладке «Файлы» указать владельца базы (имя владельца должно быть прописано в настройках входа доступа к серверу как имя входа) и во вкладке «Разрешения» внести необходимых пользователей с соответствующими правами.

2.6. Создание источника данных ODBC и взаимодействие с приложением Access

Для работы приложения необходимо определить источник данных ODBC. Данная служба присутствует в Windows и находится в Пуск, Панель управления, Администрирование, Источники данных ODBC. Для правильной настройки данной службы необходимо выбрать вкладку «Системный DNS».

Для присвоения источнику данных (базе данных SQL Server) имени и задания необходимых параметров выбирают кнопку «Добавить». Дальнейшие настройки осуществляют аналогично настройке, которая осуществлялась для базы данных SQL Server 2000.

Для того чтобы приложение MS Access 2007 взаимодействовало с базой данных SQL Server, необходимо установить связь с таблицами серверной базы данных. Для этого выбирают вкладку **Внешние данные**, раздел **Импорт**, раскрывают список **SharePoint** и выбирают пункт **База данных ODBC**.

В открывшемся диалоговом окне активируют переключатель **Создать связанную таблицу для связи с источником данных.**

Во втором диалоговом окне щелкают по кнопке **Создать,** в следующем диалоговом окне выбирают драйвер, для которого создается источник данных (например, SQL Server) и нажимают кнопку Далее.

В следующем диалоговом окне с помощью кнопки **Обзор** выбирают файл базы данных, кнопка Далее.

В списке выделяют только те таблицы, которые необходимы для работы приложения, и нажимают кнопку **ОК.**

В этом случае все данные, которые вносятся в таблицы, будут сохраняться в базе на SQL Server.

3. РАЗРАБОТКА БАЗЫ ДАННЫХ СРЕДСТВАМИ СУБД FIREBIRD

3.1. Запуск сервера Firebird

Сервер Firebird является клоном СУБД InterBase и используется для создания сетевой базы данных общего пользования.

При работе приложения с базой данных необходимо, чтобы Firebird 2.0 Server Manager был запущен. Для того чтобы проверить, запущен ли сервер БД, необходимо произвести запуск сервера, используя – Пуск/Настройка/Панель управлении/Firebird 2.0 Server Manager. При вызове диспетчера Firebird 2.0 Server Manager запустится панель управления севером Firebird Server Control, представленная на рис. 3.1.

Рис. 3.1. Панель управления сервером

Если иконка в левом углу перечеркнута красным, то сервер не запущен. Когда он запущен, иконка не перечеркнута. Запуск сервера осуществляется кнопкой «Start», а остановка

кнопкой «Stop». При этом флаг «Use the Guardian» определяет, использовать ли специальную службу "защиты" сервера Firebird и нет. Поле «Run» определяет режим запуска сервера: в режиме службы ОС (as a Service), либо в режиме простого приложения (as an application). В режиме простого приложения иконка приложения видна в панели задач. Поле «Start» определяет режим запуска самого сервера после включения компьютера: либо в автоматическом режиме (Avtomatically), либо в ручном режиме запуска (Manually).

Если сервер не запущен в автоматическом режиме после включения компьютера, то необходимо его запустить, нажав кнопку «Start», как представлено на рис. 3.2.

Рис. 3.2. Сервер во включенном режиме

3.2. Создание базы данных в Firebird

Для создания базы данных используют утилиту IBExpert. Запуск данной утилиты осуществляют с помощью меню - Пуск/ Программы/ HK-Software/ IBExpert/ IBExpert. При первом за-

пуске программа имеет иностранный интерфейс. Для настройки на русский язык необходимо выбрать пункты меню Options/ Environment Options, затем в появившемся диалоге Interface Language выбрать русский язык, в этом же диалоге Default Character Set выбрать кодировку WIN1251, чтобы в дальнейшем не указывать её при создание базы данных и в поле Default Server Version выбрать по умолчанию Firebird 2.0.

Для создания базы данных необходимо выбрать пункты меню - База данных/ Создать базу, при этом откроется окно создания базы данных (рис. 3.3).

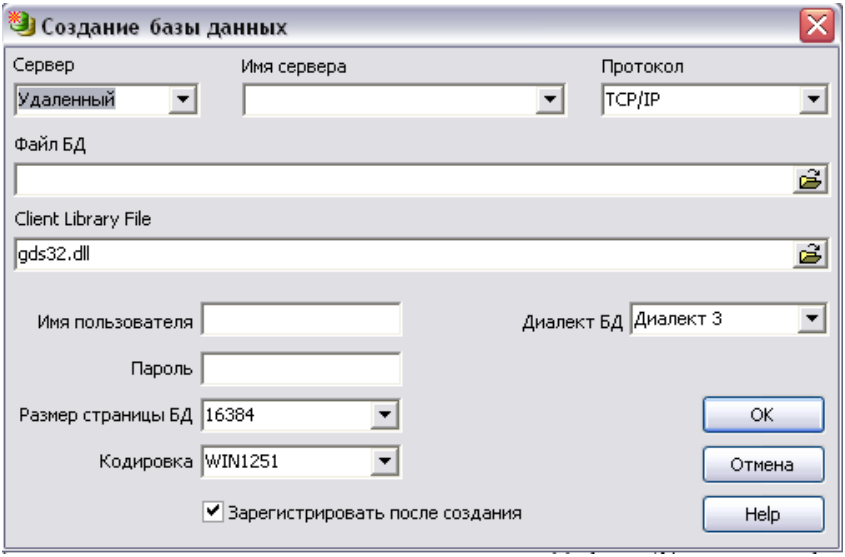

Рис. 3.3. Окно создания БД

В поле «Сервер» из раскрывающегося списка выбирают локальный (для локальной установки) или удаленный. В окне имя сервера выбирают localhost. В поле «Файл БД» вводят имя файла базы данных или выбирают с помощью кнопки поиска,

при этом определяют место, где будет расположена база данных с расширением файла «InterBase Database» (рис. 3.4).

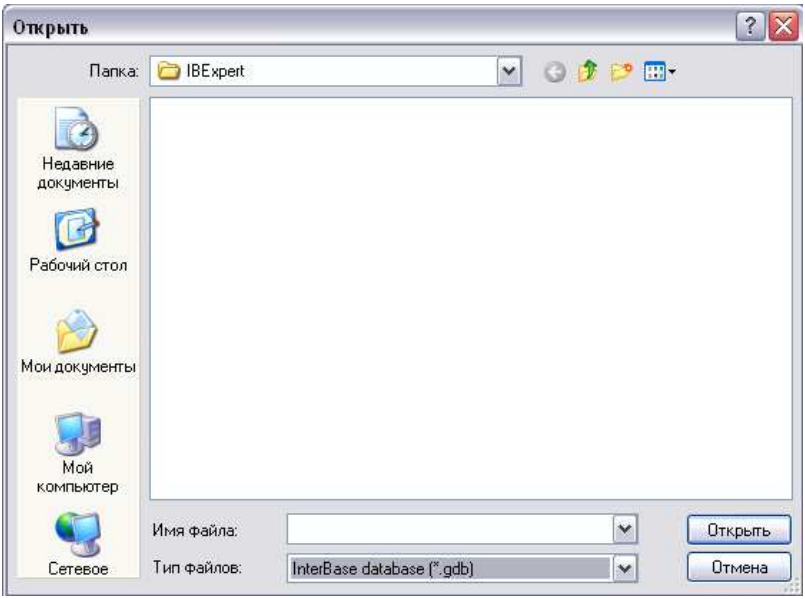

Рис. 3.4. Сохранение БД

В поле Имя пользователя (рис. 3.3) вводят имя пользователя (например, SYSDBA), а в поле Пароль - пароль (например, masterkey). Необходимо оставить нетронутым флажок «Зарегистрироваться после создания» и подтвердить создание базы данных, нажав кнопку «ОК». После подтверждения создания базы данных открывается окно регистрации, где можно изменить тип сервера, версию сервера и добавить необходимое описание базы данных, затем нажимают кнопку «Register» (рис. 3.5).

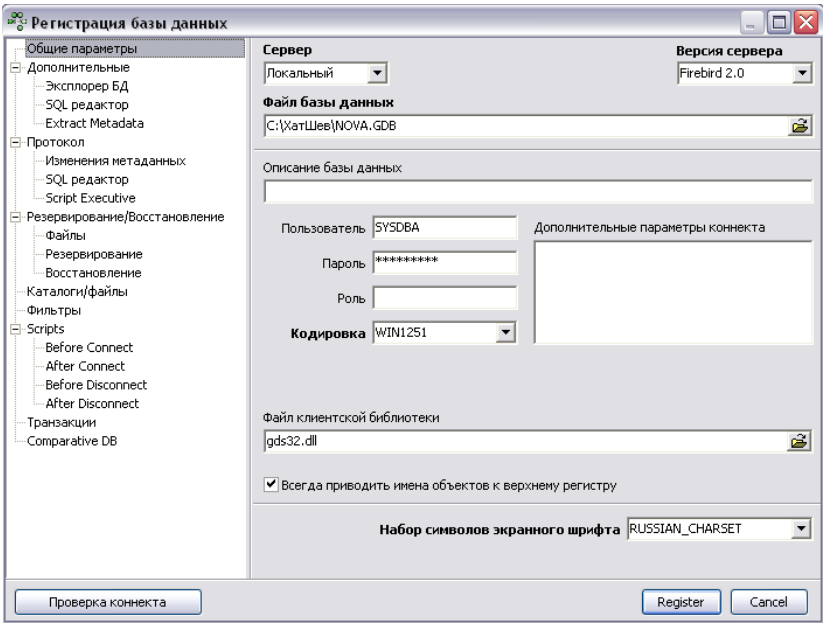

Рис. 3.5. Окно регистрации

3.3. Подключение базы данных Firebird

Для того, чтобы подключить базу данных, необходимо запустить IBExpert – Пуск/Программы/ НК-SoftWare/ IBExpert/ IBExpert.

После загрузки IBExpert слева на экране отображается дерево объектов, на дереве выделяют нужную базу данных, затем выбирают пункты меню - База данных/ Подключиться к базе. Аналогично производится и отключение базы данных (рис. 3.6).

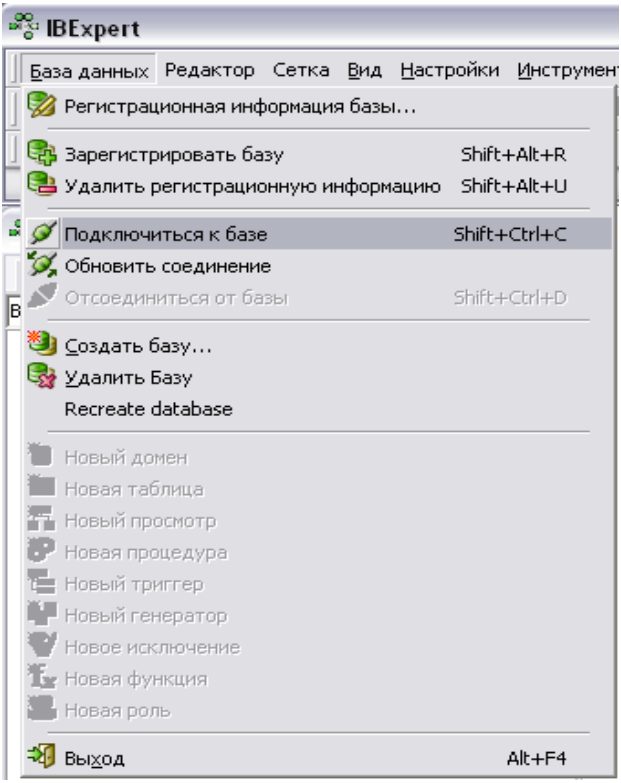

Рис. 3.6. Подключение к БД

После этого на дереве появляется подключенная база данных и ее объекты, в том числе таблицы, с которыми можно работать (рис. 3.7).

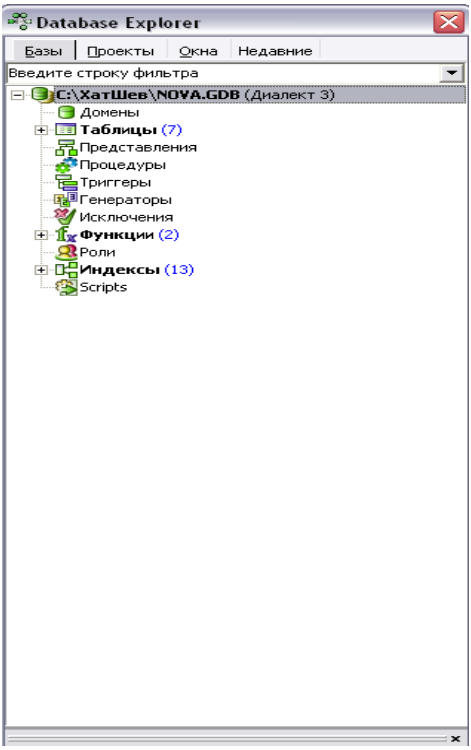

Рис. 3.7. Вид подключенной БД

3.4. Создание и редактирование таблиц Firebird

Для создания таблицы выбирают пункты меню - База данных/ Новая таблица. Справа появляется форма, в которой выбирают вкладку Поля. В эту форму вводят описания полей (рис. 3.8).

Для описания каждого поля задают следующие данные:

- имя поля, все буквы прописные английские;
- тип данных выбирают из списка;
- для текстовых полей задают длину (количество символов);
- можно указать, что поле не должно быть пустым;

- символом Ключ на стандартной панели определяют ключевое поле;

- кнопкой Компиляция сохраняют таблицу.

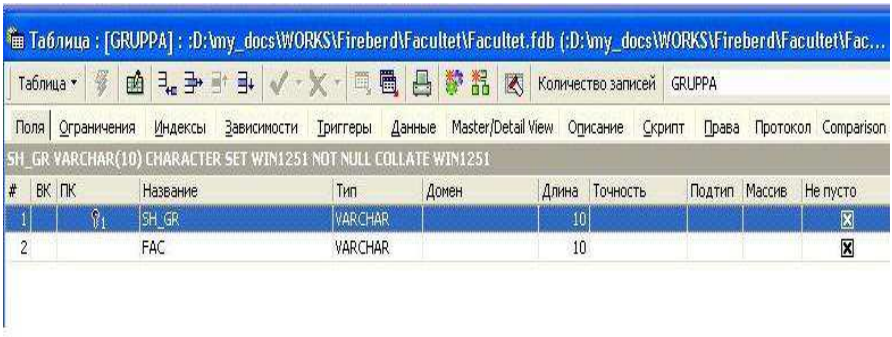

Рис. 3.8. Поля для ввода описания структуры таблицы

Для редактирования описания таблицы необходимо нужную таблицу открыть в режиме изменения таблицы (на дереве объектов вызывают на таблице контекстное меню, где выбирают пункт Изменить таблицу).

В появившемся дополнительном окне выбирают вкладку Поля.

Для изменения существующего поля на данном поле вызывают контекстное меню и выбирают пункт «Изменить поле» (рис. 3.9).

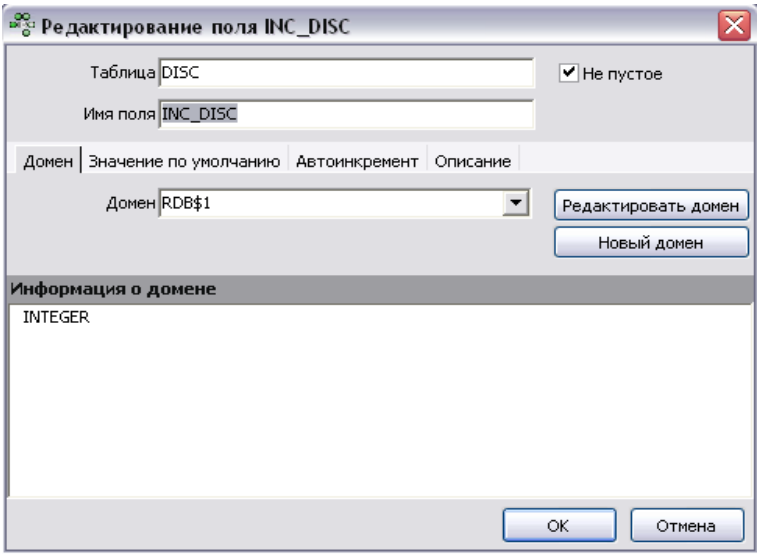

Рис. 3.9. Редактирование поля в таблице

Для удаления существующего поля на данном поле вызывают контекстное меню и выбирают пункт «Удалить поле».

Для добавления нового поля вызывают контекстное меню на любом поле и выбирают пункт «Новое поле» (рис. 3.10).

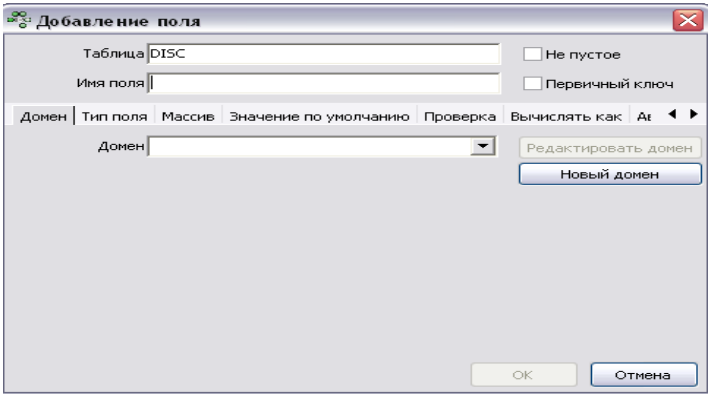

Рис. 3.10. Добавление нового поля в таблицу

В строке Имя поля вводят имя поля, затем щелкают по кнопке Тип поля. Открывается дополнительный список для выбора типа поля (рис. 3.11).

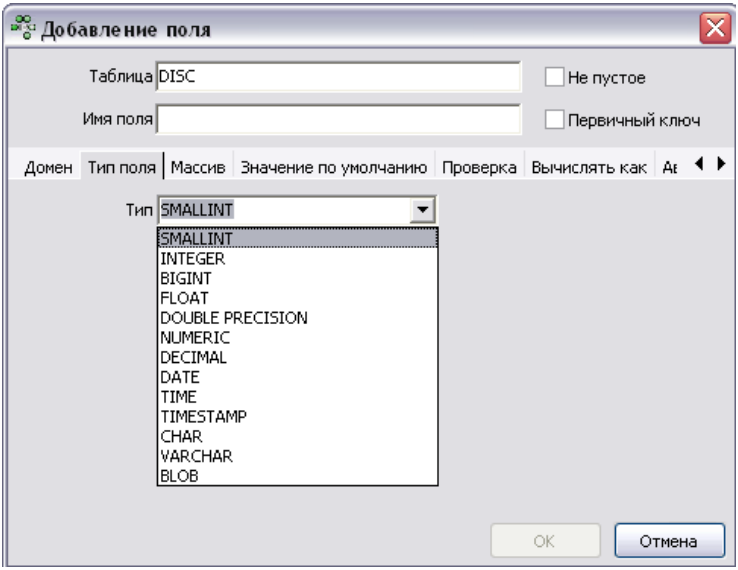

Рис. 3.11. Окно для задания нового имени поля и его типа

3.5. Связи между таблицами Firebird

Для установки связей между таблицами поступают следующим образом.

1. Открыть в режиме редактирования подчиненную таблицу (на дереве объектов на имени таблицы вызывают контекстное меню и выбирают пункт «Изменить таблицу»).

2. Выбрать для таблицы вкладку «Ограничения», а затем вкладку «Внешние ключи».

3. В открывшейся форме выбирают поле для связи (колонка На поле), имя внешней таблицы из списка, внешнее поле из списка, устанавливают правило обновления (например, Cas-

cade), правило удаления, затем щелкают по кнопке «Компиляция» для сохранения произведенных изменений.

Вид формы для установки связи с главной таблице приведен на рис. 3.12.

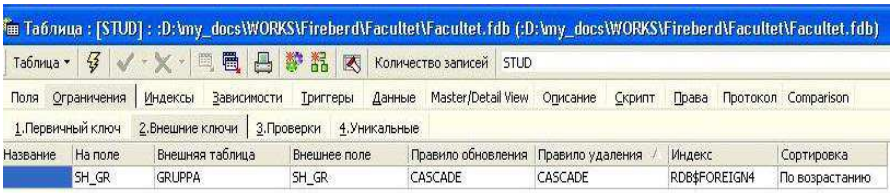

Рис. 3.12. Окно для установки связей в таблицах

3.6. Перенос базы данных на другой компьютер

Исходные файлы базы данных имеют расширение .gdb. При переносе базы данных на другой компьютер необходимо:

- скопировать файлы с названием базы данных на новый компьютер (например, на диск C:/);

- запустить сервер Firebird 2.0, выбрать меню Пуск/ Настройка/ Панель управления/ Firebird 2.0 Server Manager, нажать кнопку Старт, если она не активна, и запустить сервер;

- необходимо запустить базу данных, используя пункты меню Пуск/ Программы/ HK-Software/ IBExpert; после запуска данной программы необходимо выбрать пункт меню База данных/ Регистрация базы данных/ Выбрать файл базы данных/ Ввести имя пользователя и пароль/ Нажать кнопку Регистрация;

- выбрать зарегистрированный файл и выбрать пункты меню База данных / Подключится к базе.

Для того, чтобы база данных взаимодействовала с приложение, необходимо приложение поместить в тоже место, где

расположен файл базы данных (например, на диск C:/), в этом случае приложение будет работать корректно.

3.7. Доступ к базе данных из приложения Delphi

Для доступа к таблицам базы данных, созданным средствами СУБД Firebird 2.0 (с помощью утилиты IBExpert), поступают следующим образом.

1. На модуль данных из вкладки InterBase необходимо поставить компонент **IBDatabase**. Для него настраивают следующие свойства:

- свойство DatabaseName, выбирают путь к файлу базы данных;

- свойство Params, щелкают по кнопке … и в открывшемся окне набирают

USER_NAME=SYSDBA
PASSWORD=masterkey
Lc_ctype=WIN1251

- в свойстве LoginPrompt выставляют False.

2. На форму из той же вкладки InterBase необходимо поставить компонент **IBTransaction**. Для данного компонента в свойстве DefaultDatabase выбирают IBDatabase1.

3. На форму из той же вкладки InterBase необходимо поставить компонент **IBTable**. Для данного компонента настраивают следующие свойства:

- свойство Database, получает значение IBDatabase1;

- свойство TableName, получает значение имени таблицы (например, GRUPPA);

- свойство Active, выбирают значение true.

Дальнейшая работа с данными в таблице осуществляется стандартным образом. Например, для визуализации данных из таблицы используют компонент DataSource (в свойстве DataSet выставляют IBTable1), компонент DBGrid (в свойстве DataSource выставляют DataSource1).

4.СТРУКТУРИРОВАННЫЙ ЯЗЫК ЗАПРОСОВ SQL

4.1. История развития SQL

Язык SQL – стандартный язык запросов по работе с реляционными базами данных. Язык SQL появился после реляционной алгебры, и его прототип был разработан в конце 70-х годов в компании IBM Research. Он был реализован в первом прототипе реляционной СУБД фирмы IBM System R. В дальнейшем этот язык применялся во многих коммерческих СУБД и в силу своего широкого распространения постепенно стал стандартом «де-факто» для языков манипулирования данными в реляционной СУБД.

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

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

В 1999 году появился новый стандарт, названный SQL3. Стандарт SQL3 соответствует качественным серьезным преобразованиям. В SQL3 введены новые типы данных, при этом предполагается возможность задания сложных структурированных типов данных, которые в большой степени соответствуют объектной ориентации. Наконец, добавлен раздел, который вводит стандарты на события и триггеры. В стандарте определены возможности четкой спецификации триггеров как совокупности события и действия. В качестве действия могут выступать не только последовательность операторов SQL, но и операторы управления ходом выполнения программы.

В рамках управления транзакциями произошел возврат к старой модели транзакций, допускающей точки сохранения, и возможность указания в операторе отката ROOLBACK точек

возврата не только в начало транзакции, но и в промежуточную ранее сохраненную точку.

4.2. Структура SQL

Операторы языка SQL можно условно разделить на два подъязыка: язык определения данных (Data Definition Language – DDL) и язык манипулирования данными (Data Manipulation Language – DML). Основные операторы языка SQL представлены в табл.2. Кроме того, язык содержит операторы управления транзакциями, администрирования данных и управления курсором.

Операторы языка SQL Таблица 2

Вид	Оператор	Назначение	Действие
DDL	CREATE TABLE	Создание таблицы	Создание новой таблицы в БД
	DROP TABLE	Удаление таблицы	Удаление таблицы из БД
	ALTER TABLE	Изменение структуры таблицы	Изменение структуры существующей таблицы или ограничений целостности, задаваемые для данной таблицы
	CREATE VIEW	Создание представления	Создание виртуальной таблицы, соответствующей некоторому SQL-запросу
	DROP VIEW	Удаление представления	Удаление ранее созданного представления
	ALTER VIEW	Изменение представления	Изменение ранее созданного представления

Продолжение табл. 2

	CREATE INDEX	Создание индекса	Создание индекса для некоторой таблицы для обеспечения быстрого доступа по атрибутам, входящим в индекс
	DROP INDEX	Удаление индекса	Удаление ранее созданного индекса
DML	SELECT	Выборка записей	Формирование результирующего отношения, соответствующего запросу (оператор, заменяющий все операторы реляционной алгебры)
	UPDATE	Изменение записей	Обновление значения одного или нескольких столбцов в одной или нескольких строках, соответствующих условиям фильтрации
	INSERT	Вставка новых записей	Вставка одной строки в базовую таблицу. Допустимы модификации оператора, при которых сразу несколько строк могут быть перенесены из одной таблицы или запроса в базовую таблицу

Окончание табл. 2

	DELETE	Удаление записей	Удаление одной или нескольких строк, соответствующих условиям фильтрации, из базовой таблицы. Применение оператора согласуется с принципами поддержки целостности, поэтому этот оператор не всегда может быть выполнен корректно, даже если синтаксически он записан правильно

4.3. Оператор выбора Select

Язык запросов в SQL состоит из единственного оператора SELECT.

Синтаксис оператора SELECT имеет следующий вид:

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

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

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

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

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

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

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

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

Примеры.

Следующий оператор означает выбор всех полей из таблицы Студенты.

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

Следующий оператор означает выбор двух полей из таблицы Студенты.

```
SELECT   Номер_зачетки, Фамилия  FROM  Студенты ;
```

Следующий оператор выбирает поле Шифр_группы из таблицы Студенты, при этом повторяющиеся шифры групп удаляются. Таким образом, запрос выдает список разных шифров групп из таблицы Студенты

```
SELECT distinct Шифр_группы
From Студенты
```

4.4. Выбор полей из двух таблиц

Следующий оператор означает выбор всех полей из таблицы Студенты и таблицы Экзамены, ключевое слово WHERE задает условие соединения двух таблиц. Выбираются те записи из двух таблиц, у которых совпадают значения в поле Номер зачетки.

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

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

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

Следующий оператор означает выбор всех полей из таблицы Студенты и двух полей из таблицы Экзамены. Для таблицы Студенты выбрано второе имя С, для таблицы Экзамены выбрано второе имя Э. В данном запросе обязательно указывают условие соединения двух таблиц (в данном случае условием соединения является равенство номеров зачеток).

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

4.5. Задание условий отбора записей (WHERE)

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

В выражении условий раздела WHERE могут быть использованы следующие предикаты.

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

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

Select Продажа.* from Продажа where Количество > 10;

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

Select Экзамен.* from Экзамен where Оценка = 5;

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

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

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

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

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

Предикат вхождения в множество IN (множество) истинен тогда, когда сравниваемое значение входит в множество заданных значений. При этом множество значений может быть

задано простым перечислением или встроенным подзапросом. Одновременно существует противоположный предикат NOT IN (множество), который истинен тогда, когда сравниваемое значение не входит в заданное множество.

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

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

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

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

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

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

Для СУБД Access

Select Студенты.* from Студенты
Where Фамилия = like "С*" or Фамилия = like "М*";

Для других СУБД

Select Студенты.* from Студенты
Where Фамилия = like "С%" or Фамилия = like "М%";

Предикат сравнения с неопределенным значением IS NULL. Для выявления равенства значения некоторого атрибута неопределенному значению применяют специальные стандартные предикаты:

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

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

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

Предикат существования EXIST и не существования NOT EXIST.

В условиях поиска могут быть использованы все рассмотренные предикаты.

Примеры поиска числовых данных. Поиск числовых данных рассмотрим на примерах.

Следующий оператор выбирает все поля из таблицы Продажа, где поле Количество больше либо равно10.

Select П.* **From** Продажа П
Where Количество >= 10;

Следующий оператор выбирает все поля из таблицы Продажа, где поле Количество попадает в интервал от 10 до 100.

Select П.* **From** Продажа П
Where Количество **between** 10 **and** 100;

Примеры поиска текстовых данных. Следующий оператор выбирает все поля из таблицы Студенты, где поле Фамилия начинается с буквы П.

Select С.* **From** Студенты С
Where Фамилия **like** ‘П%’;

Следующий оператор выбирает из таблицы Студенты все поля для записей, где поле Фамилия начинается с буквы «С» или «М»:

Для СУБД Access

Select Студенты.* from Студенты
Where Фамилия = like “С*” or Фамилия = like “М*”;

Для других СУБД

Select Студенты.* from Студенты
Where Фамилия = like “С%” or Фамилия = like “М%”;

Следующий оператор выбирает все поля из таблицы Студенты, где поле Фамилия совпадает с фамилией Попова.

Select С.* **From** Студенты С
Where Фамилия = ‘Попова’;

Следующий оператор выбирает все поля из таблицы Студенты и поле Факультет из таблицы Группы, где поле Факультет совпадает со значением ФИТКБ.

Select С.*, Факультет
From Студенты С, Группы Г
Where (Факультет = ‘ФИТКБ’) **and** (С.Шифр_группы = Г.Шифр_группы);

В данном операторе обязательно надо записывать условие соединения таблиц (это условие равенства значений в поле Шифр_группы).

Примеры поиска дат. Следующий оператор выбирает все поля из таблицы Продажа, где поле Дата_продажи совпадает со значением 10.03.14.

Select П.* **From** Продажа П
Where Дата_продажи = ’10.03.14’;

Следующий оператор выбирает все поля из таблицы Продажа, где поле Дата_продажи попадает на февраль 2014.

Select П.* **From** Продажа П
Where Дата_продажи **between** ’01.02.14’ **and** ’28.02.14’;

Следующий оператор выбирает все поля из таблицы Продажа, где дата продажи меньше 20.02.14.

Select П.* **From** Продажа П
Where Дата_продажи < ’20.02.14’;

4.6. Запрос с вычисляемым полем

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

Общий вид оператора выбора с вычисляемым полем следующий:

Select имена полей,

формула вычислений **as** имя вычисляемого поля

From имя таблицы;

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

Select П.*, Цена * Количество **as** Стоимость_покупки **from** Продажа П;

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

Select Т.Название, Т.Цена, П.Чек, П.Дата_продажи, П.Количество, Т.Цена * П.Количество as Стоимость_покупки

from Товары Т, Продажа П

where Т.Код = П.Код;

При выборе полей из разных таблиц необходимо:

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

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

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

Select K.*, P.*, K.Cena * P.Kol as Stoim_post

From Knigi K, Postavka P

Where K.Shifr = P.Shifr;

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

4.7. Запрос с группировкой и применение агрегатных функций (GROUP BY)

Общий вид запроса с группировкой:

Select поля группировки,

функция(поле для вычислений по группе записей) **as** имя вычисляемого поля

From имя таблицы

Group by поля группировки;

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

В SQL добавлены дополнительные функции, которые позволяют вычислять обобщенные групповые значения. Для применения агрегатных функций предполагается предварительная операция группировки. При группировке все множество кортежей отношения разбивается на группы, в которых объединяются кортежи, имеющие одинаковые значения атрибутов, которые заданы в списке группировки.

Список агрегатных функций представлен в табл. 3.

Таблица 3

Агрегатные функции

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

Агрегатные функции применяются подобно именам полей в операторе SELECT, но они используют имя поля как аргумент. С функциями SUM и AVG могут использоваться только числовые поля. С функциями COUNT, MAX, MIN могут использоваться как числовые, так и символьные поля. При использовании с символьными полями MAX и MIN будут транслировать их в эквивалент ASCII кода и обрабатывать в алфавитном порядке.

Рассмотрим несколько примеров на применение группировки и агрегатных функций.

Пусть даны следующие таблицы:

Дисциплины (Код_дисциплины, Название_дисциплины);

Группы (Шифр_группы, Факультет, Специальность, Курс);

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

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

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

Пример 1. Определить количество студентов в каждой группе.

Select Шифр_группы, count(Номер_зачетки) As Кол_студентов
From Студенты Group by Шифр_группы;

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

Select Факультет, С.Шифр_группы,

count(Номер_зачетки) As Кол_студентов
From Группы Г, Студенты С
Where Г.Шифр_группы = С.Шифр_группы
Group by Факультет, С.Шифр_группы;

Пример 3. Определить количество групп каждой специальности и курса.

Select Специальность, Курс, count(Шифр_группы)
As Кол_групп
From Группы
Group by Специальность, Курс;

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

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

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

Имеются две таблицы, связанные по полю Филиал.

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

Q (Филиал, Город)

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

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

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

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

```
SELECT Филиал, SUM(Остаток)
FROM F, Q
WHERE F.Филиал = Q.Филиал
GROUP BY Филиал
HAVING Город IN («Воронеж», «Липецк», «Курск»);
```

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

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

Select Шифр_группы, count(Номер_зачетки)

As Кол_студентов From Студенты
Group by Шифр_группы
Having count(Номер_зачетки) > 20;

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

Select Специальность, С.Шифр_группы,
count(Номер_зачетки) As Кол_студентов
From Студенты С, Группы Г
Where С.Шифр_группы = Г.Шифр_группы
Group by Специальность, С.Шифр_группы
Having Специальность = "ВМ";

4.8. Раздел ORDER BY и ключевое слово TOP

Раздел определяет порядок сортировки записей, включенных в выборку:

Select имена поле **From** имя таблицы
ORDER BY поле1 [**ASC** / **DESC**]
[, поле2 [**ASC** / **DESC**] [,…]]

Особенности:

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

Примеры.

Отсортировать записи в таблице Студенты по полю ФИО в алфавитном порядке:

```
SELECT C.*
FROM Студенты C
ORDER BY ФИО;
```

Отсортировать записи в таблице Экзамены в порядке убывания значений в поле Оценка:

```
SELECT Э.*
FROM Экзамены Э
ORDER BY Оценка DESC;
```

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

Например, выбрать из таблицы Студенты (Номер_зачетки, Шифр_группы, ФИО, Год_выпуска, Средний_балл) 5 лучших студентов выпуска 2013 года:

```
SELECT  TOP  5 ФИО, Шифр_группы
FROM Студенты
WHERE  Год_выпуска = 2013
ORDER  BY Средний_балл  DESC;
```

Число, используемое в предикате TOP, должно быть целым без знака. Без директивы ORDER BY в выборку попали бы любые 5 студентов выпуска 2013 года. Предикат TOP не разделяет записи, имеющие одинаковые значения при упорядочивании. Это значит, если 5-й, 6-й, 7-й студенты имеют одинаковый средний балл, то в выборке будет не 5, а 7 записей.

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

Например, выбрать десять процентов записей выпуска 2013 года из таблицы Студенты:

```
SELECT TOP 10 PERCENT  ФИО, Шифр_группы
FROM Студенты
WHERE Год_выпуска = 2013
ORDER  BY Средний_балл DESC;
```

4.9. Перекрестные запросы

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

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

Пример 1. Дана таблица Продажа_бензина (Чек, Марка, Дата_продажи, Оператор, Количество). Определить суммарную

продажу каждой марки бензина за каждый день. Марки бензина вывести в столбцах, даты – в строках, суммарную продажу – на пересечении строк и столбцов.

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

Пример 2. Даны таблицы Продажа_бензина (Чек, Марка, Дата_продажи, Номер_сотрудника, Количество, Цена) и Сотрудники (Номер_сотрудника, Фамилия, Имя, Отчество, Должность, Оклад). Определить суммарную стоимостную продажу бензина каждым сотрудником за каждый день. В строках вывести даты, в столбцах – фамилии, на пересечении строки и столбца – суммарную стоимостную продажу.

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

ЗАКЛЮЧЕНИЕ

Информационные системы, применяющие распределенную обработку данных, находят широкое применение во всех сферах деятельности человека.

Данное пособие содержит описание двух основных технологий работы с распределенными данными:

- распределенная обработка данных на основе применения централизованной базы данных;
- распределенные базы данных.

Пособие охватывает широкий круг вопросов, связанных с распределенной обработкой данных на основе централизованной базы данных:

- двухуровневые модели данных и распределение выполняемых функций между сервером и клиентом;
- трехуровневые системы обработки данных.

В пособии рассмотрены организация работы распределенных баз данных, проблемы проектирования и эксплуатации.

Вторая глава рассматривает особенности разработки централизованных баз данных с применением СУБД SQL Server и взаимодействие с ней приложений, разработанных в СУБД Access.

Третья глава посвящена созданию централизованной базы данных с использованием СУБД Firebird, а также рассматривается доступ к базе данных из приложения, разработанного в среде программирования Delphi.

В четвертой главе рассматриваются особенности применения языка SQL для реализации запросов на выборку данных, с вычисляемыми полями, с групповыми вычислениями, реализацию параметрических и перекрестных запросов. Приведенные примеры позволят практически освоить создание различных запросов на выборку данных.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Хомоненко, А.Д. Базы данных [Текст] / А.Д. Хомоненко, В.М. Цыганков, М.Г. Мальцев. - СПб.: КОРОНА принт, 2004. – 416 с.

2. Советов, Б.Я. Базы данных: теория и практика [Текст] / Б.Я. Советов, В.В. Цехановский, В.Д. Чертовской. М.: Издательство Юрайт, 2012. – 464 с.

3. Карпова, Т.С. Базы данных: модели, разработка, реализация [Текст] / Т.С. Карпова. – СПб.: Питер, 2001. – 304 с.

4. Саак, А.Э. Информационные технологии управления [Текст] / А.Э. Саак, Е.В. Пахомов. В.Н. Тюшняков. – СПб.: Питер, 2008. – 320 с.

5. Волоха, А.В. Microsoft SQL Server 2005. Новые возможности [Текст] / А.В. Волоха. СПб.: Питер, 2006. – 845 с.

6. Нильсен, П. SQL Server 2005. Библия пользователя [Текст]: пер. с англ. / П. Нильсен. - М.: Вильямс, 2008. – 456 с.

7. Диго, С.М. Access [Текст]: учеб. пособие / С.М. Диго. М.: Прогресс, 2006. – 326 с.

8. Ковязин, А. Мир InterBase. Архитектура, администрирование и разработка приложений баз данных в InterBase/Firebird/Yaffil [Текст] / А. Ковязин, С. Востриков. – М.: КУДИЦ-ОБРАЗ, 2006. – 496 с.

ОГЛАВЛЕНИЕ

Учебное издание

Сергеева Татьяна Ивановна
Сергеев Михаил Юрьевич

РАСПРЕДЕЛЕННАЯ ОБРАБОТКА ДАННЫХ

В авторской редакции

Компьютерная верстка Т.И. Сергеевой

Подписано к изданию 23.10.2014.

Объем данных 784 Кб

ФГБОУ ВПО «Воронежский государственный технический университет»

394026 Воронеж, Московский просп., 14

96