

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»

М. В. Додонов, Е. В. Сопченко

Распределенная обработка данных в современных СУБД

Электронное учебное пособие

САМАРА

2010

Авторы: Додонов Михаил Витальевич,

Сопченко Елена Вильевна

Рецензент: к.т.н, доц. А.В. Авсиевич

В данном электронном учебном пособии рассмотрены основные вопросы распределенной обработки данных в современных системах управления базами данных. Даны краткие теоретические сведения по основным разделам, соответствующим рабочей программе. Основные понятия проиллюстрированы достаточным количеством примеров. Рассмотрены различные архитектуры распределенной обработки данных и перспективы развития этих методов применительно к современным СУБД.

Учебное пособие рекомендуется для магистрантов по курсу лекций «Распределенная обработка данных в современных СУБД» в рамках магистерской программы 010300.68 «Фундаментальная информатика и информационные технологии» и может быть полезно при выполнении квалификационных работ и подготовке к экзаменам.

Подготовлено на кафедре программных систем.

© Самарский государственный
аэрокосмический университет, 2010

Содержание

Введение.....	4
1 История развития СУБД.....	6
1.1 Файловые системы.....	7
1.2 Иерархические базы данных	10
1.3 Сетевые базы данных	11
1.4 Реляционные базы данных.....	13
2 Современные технологии обработки данных	24
2.1 Основные компоненты	24
2.2 Режимы обработки данных.....	27
2.3 Способы обработки данных.....	29
3 Распределенная обработка данных и распределенные базы данных	32
3.1 Системы распределённой обработки данных.....	36
3.2 Системы распределенных баз данных.....	39
3.3 Системы управления распределёнными базами данных.....	41
4 Модели распределенной обработки данных в современных СУБД.....	43
4.1 Двухзвенная модель	47
4.2 Модель сервера приложений.....	54
4.3 Публикация баз данных в Internet.....	56
4.4 Обеспечение безопасности данных	65
5 Модели распределенных баз данных	75
5.1 Распределенные и параллельные базы данных.	80
5.2 Однородные и неоднородные распределенные СУБД	87
5.3 Вертикальная и горизонтальная фрагментация данных в РБД.....	91
5.4 Репликация (тиражирование) данных в РБД	95
6 Перспективы распределенной обработки данных в современных СУБД	98
Список используемых источников.....	106

Введение

Требования, предъявляемые к современным информационным системам (ИС), и развитие новых информационных технологий (НИТ) приводит к постоянному росту сложности разработки и последующей эксплуатации таких систем. Современные информационные системы характеризуются следующими особенностями:

- сложность описания, требующая тщательного моделирования и анализа данных и процессов;
- наличие совокупности тесно взаимодействующих подсистем, имеющих свои локальные задачи и цели функционирования;
- необходимость интеграции существующих и вновь разрабатываемых приложений;
- функционирование в неоднородной среде на нескольких аппаратных платформах;
- разнородность инструментальных средств проектирования и разработки ИС.

Анализ современных тенденций развития НИТ свидетельствует об усложнении используемых в них структур данных при одновременном увеличении количества хранимой и обрабатываемой информации. Для обработки все возрастающего объема информации требуется новое аппаратное и соответствующее программное обеспечение, поскольку существующее уже не справляется с поставленными задачами.

Таким образом, появилась идея объединения вычислительных мощностей двух и более компьютеров для решения сложных, непосильных для каждого из них в отдельности задач. Появление локальных сетей привело к развитию новой области разработки программного обеспечения - созданию распределенных приложений. Заинтересованность в таких приложениях появилась, прежде всего, у крупных компаний, структура бизнеса которых требовала подобных решений. Именно на этапе создания корпоративных распределенных приложений были сформированы основные требования и разработаны основные архитектуры подобных систем, используемые и в настоящее время.

Постепенно мэйнфреймы и терминалы эволюционировали в направлении архитектуры «клиент-сервер», которая по существу была первым вариантом распределенной архитектуры, т. е. двухуровневой распределенной системой. Ведь именно в приложениях «клиент-сервер» часть вычислительных операций и «бизнес-логики» были перенесены на сторону клиента, что, собственно, и легло в основу этого подхода к распределенной обработке данных.

Именно в этот период стало очевидно, что основными преимуществами распределенных приложений являются:

- хорошая масштабируемость;
- возможность управления нагрузкой;
- глобальность.

Под масштабируемостью понимается возможность увеличения вычислительной мощности распределенного приложения без изменения его структуры. Возможность управления нагрузкой позволяет промежуточным уровням распределенного приложения управлять потоками запросов пользователей и перенаправлять их менее загруженным серверам для обработки. Распределенная структура позволяет следовать пространственному распределению бизнес-процессов и создавать клиентские рабочие места в наиболее удобных точках, что дает возможность решения глобальных задач.

Появление глобальной сети Internet сыграло решающую роль в развитии распределенных вычислений и сделало эту область разработки программного обеспечения предметом интереса профессиональных программистов. Сегодня Internet существенно расширяет возможности применения распределенных приложений, позволяя подключать удаленных пользователей и делая функции приложения доступными повсеместно.

С ростом объема обрабатываемой информации появилась необходимость распределенной обработки данных с использованием современных систем управления базами данных (СУБД). Данное направление, в настоящее время, активно развивается и требует более детального изучения.

1 История развития СУБД

Использование современной вычислительной техники и новых информационных технологий исторически развивалось в двух основных направлениях: численные расчеты и обработка больших объемов информации. Применение вычислительной техники для выполнения расчетов способствовало развитию численных методов решения сложных математических задач. Кроме того, данное направление способствовало появлению различных языков программирования, ориентированных на разные предметные области. Также появление новых сред разработки программ и систем автоматизированного проектирования обусловлено растущей сложностью разрабатываемого программного обеспечения.

Второе основное направление использования средств вычислительной техники - хранение и обработка больших объемов информации. Это направление широко используется в современных информационных системах. В самом широком смысле информационная система представляет собой программный комплекс, функции которого состоят в поддержке надежного хранения информации в памяти компьютера, выполнении специфических для данного приложения преобразований информации и/или вычислений, предоставлении пользователям удобного и легко осваиваемого интерфейса. Обычно объемы информации, с которыми приходится иметь дело таким системам, достаточно велики, а сама информация имеет достаточно сложную структуру. Классическими примерами информационных систем являются банковские системы, системы резервирования авиационных или железнодорожных билетов, мест в гостиницах и т.д.

Второе направление возникло несколько позже первого. Это связано с тем, что на заре вычислительной техники компьютеры обладали ограниченными объемами оперативной и внешней памяти. С появлением внешних накопителей с прямым методом доступа началась история развития

систем управления данными во внешней памяти. До этого каждая прикладная программа, которой требовалось хранить данные во внешней памяти, сама определяла расположение каждой порции данных. Такой режим работы не позволяет или очень затрудняет поддержание на одном внешнем носителе нескольких баз данных и вызывает проблемы структуризации данных во внешней памяти.

С другой стороны, для информационных систем, в которых потребность в текущих данных определяется пользователем, наличие внешних носителей информации большого объема недостаточно. Одним из естественных требований к таким системам является средняя скорость выполнения операций. Решение проблем хранения и доступа к данным большого объема привело к появлению специализированного программного обеспечения – систем управления базами данных (СУБД).

Можно выделить следующие этапы развития СУБД:

1. Файловые системы.
2. Иерархические базы данных.
3. Сетевые базы данных.
4. Реляционные базы данных.

Рассмотрим этапы развития баз данных более подробно.

1.1 Файловые системы

Историческим шагом в развитии СУБД явился переход к использованию централизованных систем управления файлами. С точки зрения прикладной программы файл - это именованная область внешней памяти для хранения данных заданного объема. Правила именования файлов, способ доступа к данным, хранящимся в файле, и структура этих данных зависят от конкретной системы управления файлами и, возможно, от типа файла. Система управления файлами берет на себя распределение внешней памяти, отображение имен

файлов в соответствующие адреса во внешней памяти и обеспечение доступа к данным.

Файлы применяются для хранения текстовых данных: документов, текстов программ и т.д. Такие файлы обычно образуются и модифицируются с помощью различных текстовых редакторов. Структура текстовых файлов обычно очень проста: это либо последовательность записей, содержащих строки текста, либо последовательность байтов, среди которых встречаются специальные символы (например, символы конца строки).

Файлы с текстами программ используются как входные тексты компиляторов, которые в свою очередь формируют файлы, содержащие объектные модули. С точки зрения файловой системы, объектные файлы также обладают очень простой структурой - последовательность записей или байтов. Система программирования накладывает на эту структуру более сложную и специфичную для этой системы структуру объектного модуля. Подчеркнем, что логическая структура объектного модуля неизвестна файловой системе, эта структура поддерживается программами системы программирования.

Аналогично обстоит дело с файлами, формируемыми редакторами связей и содержащими образы выполняемых программ. Логическая структура таких файлов остается известной только редактору связей и загрузчику - программе операционной системы. Примерно такая же ситуация с файлами, содержащими графическую и звуковую информацию.

Файловые системы обычно обеспечивают хранение слабо структурированной информации, оставляя дальнейшую структуризацию прикладным программам. В перечисленных выше случаях использования файлов обоснованно, потому что при разработке любой новой прикладной системы опираясь на простые, стандартные и сравнительно дешевые средства файловой системы можно реализовать те структуры хранения, которые наиболее естественно соответствуют специфике данной прикладной области.

Использование файловых систем для хранения обрабатываемых данных в современных информационных системах, как правило, неэффективно. Эти системы главным образом ориентированы на хранение, выбор и модификацию постоянно существующей информации. Структура информации зачастую очень сложна, и хотя структуры данных различны в разных информационных системах, между ними часто бывает много общего. На начальном этапе использования вычислительной техники для управления информацией проблемы структуризации данных решались индивидуально в каждой информационной системе. Производились необходимые надстройки над файловыми системами (библиотеки программ), подобно тому, как это делается в компиляторах, редакторах и т.д.

Но поскольку информационные системы требуют сложных структур данных, эти дополнительные индивидуальные средства управления данными являлись существенной частью информационных систем и практически повторялись от одной системы к другой. Стремление выделить и обобщить общую часть информационных систем, ответственную за управление сложно структурированными данными, явилось первой побудительной причиной создания СУБД. Очень скоро стало понятно, что невозможно обойтись общей библиотекой программ, реализующей над стандартной базовой файловой системой более сложные методы хранения данных.

СУБД решают множество проблем, которые затруднительно или вообще невозможно решить при использовании файловых систем. При этом существуют приложения, для которых вполне достаточно файлов; приложения, для которых необходимо решать, какой уровень работы с данными во внешней памяти для них требуется, и приложения, для которых нужно использование СУБД.

1.2 Иерархические базы данных

Иерархическая БД состоит из упорядоченного набора деревьев; более точно, из упорядоченного набора нескольких экземпляров одного типа дерева.

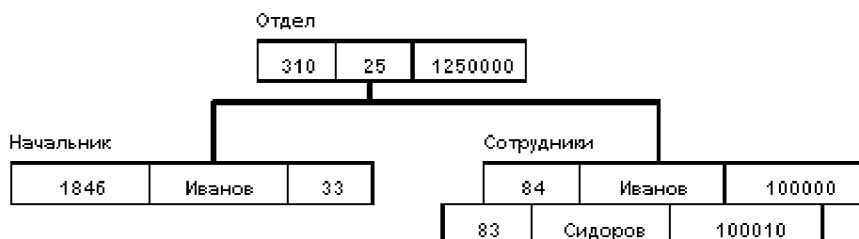
Тип дерева состоит из одного "корневого" типа записи и упорядоченного набора из нуля или более типов поддеревьев (каждое из которых является некоторым типом дерева). Тип дерева в целом представляет собой иерархически организованный набор типов записи.

Пример типа дерева (схемы иерархической БД):



Здесь Отдел является предком для Начальник и Сотрудники, а Начальник и Сотрудники - потомки Отдел. Между типами записи поддерживаются связи.

База данных с такой схемой могла бы выглядеть следующим образом (здесь представлен один экземпляр дерева):



Все экземпляры данного типа потомка с общим экземпляром типа предка называются близнецами. Для БД определен полный порядок обхода - сверху-вниз, слева-направо.

Примерами типичных операторов манипулирования иерархически организованными данными могут быть следующие:

- найти указанное дерево БД (например, отдел 310);
- перейти от одного дерева к другому;

- перейти от одной записи к другой внутри дерева (например, от отдела - к первому сотруднику);
- перейти от одной записи к другой в порядке обхода иерархии;
- вставить новую запись в указанную позицию;
- удалить текущую запись.

Автоматически поддерживается целостность ссылок между предками и потомками. Основное правило: никакой потомок не может существовать без своего родителя. Заметим, что аналогичное поддержание целостности по ссылкам между записями, не входящими в одну иерархию, не поддерживается.

В иерархических системах поддерживалась некоторая форма представлений БД на основе ограничения иерархии. Примером представления приведенной выше БД может быть иерархия:



1.3 Сетевые базы данных

Сетевой подход к организации данных является расширением иерархического. В иерархических структурах запись-потомок должна иметь в точности одного предка; в сетевой структуре данных потомок может иметь любое число предков.

Сетевая БД состоит из набора записей и набора связей между этими записями, а если говорить более точно, из набора экземпляров каждого типа из заданного в схеме БД набора типов записи и набора экземпляров каждого типа из заданного набора типов связи.

Тип связи определяется для двух типов записи: предка и потомка. Экземпляр типа связи состоит из одного экземпляра типа записи предка и

упорядоченного набора экземпляров типа записи потомка. Для данного типа связи L с типом записи предка P и типом записи потомка C должны выполняться следующие два условия:

- каждый экземпляр типа P является предком только в одном экземпляре L;
- каждый экземпляр C является потомком не более, чем в одном экземпляре L.

На формирование типов связи не накладываются особые ограничения; возможны, например, следующие ситуации:

1. Тип записи потомка в одном типе связи L1 может быть типом записи предка в другом типе связи L2 (как в иерархии).
2. Данный тип записи P может быть типом записи предка в любом числе типов связи.
3. Данный тип записи P может быть типом записи потомка в любом числе типов связи.
4. Может существовать любое число типов связи с одним и тем же типом записи предка и одним и тем же типом записи потомка; и если L1 и L2 - два типа связи с одним и тем же типом записи предка P и одним и тем же типом записи потомка C, то правила, по которым образуется родство, в разных связях могут различаться.
5. Типы записи X и Y могут быть предком и потомком в одной связи и потомком и предком - в другой.
6. Предок и потомок могут быть одного типа записи.

Простой пример сетевой схемы БД:



Примерный набор операций может быть следующим:

- найти конкретную запись в наборе однотипных записей (инженера Сидорова);
- перейти от предка к первому потомку по некоторой связи (к первому сотруднику отдела 310);
- перейти к следующему потомку в некоторой связи (от Сидорова к Иванову);
- перейти от потомка к предку по некоторой связи (найти отдел Сидорова);
- создать новую запись;
- уничтожить запись;
- модифицировать запись;
- включить в связь;
- исключить из связи;
- переставить в другую связь и т.д.

Сильные места ранних СУБД:

- развитые средства управления данными во внешней памяти на низком уровне;
- возможность построения вручную эффективных прикладных систем;
- возможность экономии памяти за счет разделения подобъектов (в сетевых системах).

Недостатки:

- слишком сложно пользоваться;
- фактически необходимы знания о физической организации;
- прикладные системы зависят от этой организации;
- их логика перегружена деталями организации доступа к БД.

1.4 Реляционные базы данных

Реляционный подход является наиболее распространенным в настоящее время, хотя наряду с общепризнанными достоинствами обладает и рядом недостатков. К числу достоинств реляционного подхода можно отнести:

- наличие небольшого набора абстракций, которые позволяют сравнительно просто моделировать большую часть распространенных предметных областей и допускают точные формальные определения, оставаясь интуитивно понятными;
- наличие простого и в то же время мощного математического аппарата, опирающегося главным образом на теорию множеств и математическую логику и обеспечивающего теоретический базис реляционного подхода к организации баз данных;
- возможность ненавигационного манипулирования данными без необходимости знания конкретной физической организации баз данных во внешней памяти.

Реляционные системы далеко не сразу получили широкое распространение. В то время как основные теоретические результаты в этой области были получены еще в 70-х годах, и тогда же появились первые прототипы реляционных СУБД, долгое время считалось невозможным добиться эффективной реализации таких систем. Однако отмеченные выше преимущества и постепенное накопление методов и алгоритмов организации реляционных баз данных и управления ими привели к тому, что уже в середине 80-х годов реляционные системы практически вытеснили с мирового рынка ранние СУБД.

Реляционная модель данных основывается на математических принципах, вытекающих непосредственно из теории множеств и логики предикатов. Эти принципы впервые были применены в области моделирования данных в конце 1960-х гг. доктором Е.Ф. Коддом, в то время работавшим в IBM, а впервые опубликованы - в 1970 г.

Техническая статья "Реляционная модель данных для больших разделяемых банков данных" доктора Е.Ф. Кодда, опубликованная в 1970 г., является родоначальницей современной теории реляционных БД. Доктор Кодд

определил 13 правил реляционной модели (которые называют 12 правилами Кодда).

12 правил Кодда:

0. Реляционная СУБД должна быть способна полностью управлять базой данных через ее реляционные возможности.
1. Информационное правило - вся информация в реляционной БД (включая имена таблиц и столбцов) должна определяться строго как значения в таблицах.
2. Гарантированный доступ - любое значение в реляционной БД должно быть гарантированно доступно для использования через комбинацию имени таблицы, значения первичного ключа и имени столбца
3. Поддержка пустых значений (null value) - СУБД должна уметь работать с пустыми значениями (неизвестными или неиспользованными значениями), в отличие от значений по умолчанию и независимо для любых доменов.
4. Онлайн-реляционный каталог - описание БД и ее содержания должны быть представлены на логическом уровне как таблицы, к которым можно применять запросы, используя язык базы данных.
5. Исчерпывающий язык управления данными - по крайней мере, один из поддерживаемых языков должен иметь четко определенный синтаксис и быть всеобъемлющим. Он должен поддерживать описание структуры данных и манипулирование ими, правила целостности, авторизацию и транзакции.
6. Правило обновления представлений (views) - все представления, теоретически обновляемые, могут быть обновлены через систему.
7. Вставка, обновление и удаление - СУБД поддерживает не только запрос на отбор данных, но и вставку, обновление и удаление

8. Физическая независимость данных - на программы-приложения и специальные программы логически не влияют изменения физических методов доступа к данным и структур хранилищ данных.
9. Логическая независимость данных - на программы-приложения и специальные программы логически не влияют, в пределах разумного, изменения структур таблиц.
10. Независимость целостности - язык БД должен быть способен определять правила целостности. Они должны сохраняться в онлайн-овом справочнике, и не должно существовать способа их обойти.
11. Независимость распределения - на программы-приложения и специальные программы логически не влияет, первый раз используются данные или повторно.
12. Неподрывность - невозможность обойти правила целостности, определенные через язык базы данных, использованием языков низкого уровня

Кодд предложил применение реляционной алгебры в СУРБД, для расчленения данных в связанные наборы. Он организовал свою систему БД вокруг концепции, основанной на наборах данных.

В реляционной модели данные разбиваются на наборы, которые составляют табличную структуру. Эта структура таблиц состоит из индивидуальных элементов данных, называемых полями. Одиночный набор или группа полей известна как запись. Модель данных, или концептуальное описание предметной области - самый абстрактный уровень проектирования баз данных.

С точки зрения теории реляционных БД, основные принципы реляционной модели на концептуальном уровне можно сформулировать следующим образом:

- все данные представляются в виде упорядоченной структуры, определенной в виде строк и столбцов и называемой отношением;
- все значения являются скалярами. Это означает, что для любой строки и столбца любого отношения существует одно и только одно значение;
- все операции выполняются над целым отношением, и результатом их выполнения также является целое отношение. Этот принцип называется замыканием

Формулируя принципы реляционной модели, доктор Кодд выбрал термин "отношение" (relation), потому что, по его мнению, этот термин однозначен (в то время как, например, термин "таблица" имеет множество различных видов - таблица в тексте, электронная таблица и пр.). Весьма распространено следующее заблуждение: реляционная модель названа так потому, что она определяет связи между таблицами. На самом деле, название этой модели происходит от отношений (таблиц базы данных), лежащих в ее основе.

Основными понятиями реляционных баз данных являются тип данных, домен, атрибут, кортеж, первичный ключ и отношение. Реляционная база данных - это набор отношений, имена которых совпадают с именами схем отношений в схеме БД.

Основные структурные понятия реляционной модели данных (если не считать понятия домена) имеют очень простую интуитивную интерпретацию, хотя в теории реляционных БД все они определяются абсолютно формально и точно.

Каждая строка, содержащая данные, называется кортежем, каждый столбец отношения называется атрибутом (на уровне практической работы с современными реляционными БД используются термины "запись" и "поле").

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

Сущность - некоторый обособленный объект или событие, информацию о котором необходимо сохранять в базе данных, имеющий определенный набор свойств - атрибутов. Сущности могут быть как физические (реально существующие объекты: например, СТУДЕНТ, атрибуты - № зачетной книжки, фамилия, его факультет, специальность, № группы и т.д.), так и абстрактные (например, ЭКЗАМЕН, атрибуты - дисциплина, дата, преподаватель, аудитория и пр.). Для сущностей различают ее тип и экземпляр. Тип характеризуется именем и списком свойств, а экземпляр - конкретными значениями свойств.

Атрибуты сущности бывают:

1. Идентифицирующие и описательные. Идентифицирующие атрибуты имеют уникальное значение для сущностей данного типа и являются потенциальными ключами. Они позволяют однозначно распознавать экземпляры сущности. Из потенциальных ключей выбирается один первичный ключ (ПК). В качестве ПК обычно выбирается потенциальный ключ, по которому чаще происходит обращение к экземплярам записи. ПК должен включать в свой состав минимально необходимое для идентификации количество атрибутов. Остальные атрибуты называются описательными.
2. Простые и составные. Простой атрибут состоит из одного компонента, его значение неделимо. Составной атрибут является комбинацией нескольких компонентов, возможно, принадлежащих разным типам данных (например, адрес). Решение о том, использовать составной атрибут или разбивать его на компоненты, зависит от особенностей процессов его использования и может быть связано с обеспечением высокой скорости работы с большими базами данных.
3. Однозначные и многозначные - могут иметь соответственно одно или много значений для каждого экземпляра сущности.

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

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

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

Связи - на концептуальном уровне - представляют собой простые ассоциации между сущностями. Например, утверждение "Покупатели приобретают продукты" указывает, что между сущностями "Покупатели" и "Продукты" существует связь, и такие сущности называются участниками этой связи.

Существует несколько типов связей между двумя сущностями: это связи "один к одному", "один ко многим" и "многие ко многим".

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

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

В настоящее время основным предметом критики реляционной модели является присущая ей некоторая ограниченность (прямое следствие простоты) при использовании в так называемых нетрадиционных областях (наиболее распространенными примерами являются системы автоматизации проектирования), в которых требуются предельно сложные структуры данных. Еще одним часто отмечаемым недостатком реляционных баз данных является невозможность адекватного отражения семантики предметной области. Другими словами, возможности представления знаний о семантической специфике предметной области в реляционных системах очень ограничены. Современные исследования в области постреляционных систем главным образом посвящены именно устранению этих недостатков.

Заключение по главе 1

Как уже упоминалось ранее, предшественницами СУБД были файловые системы. Однако появление СУБД не привело к их полному исчезновению: для выполнения некоторых специализированных задач подобные файловые системы используются до сих пор. Кроме того, файловые системы могут использоваться также СУБД для решения задач хранения данных и доступа к ним.

В середине 60-х годов корпорация IBM совместно с фирмой NAA (North American Aviation, в настоящее время - Rockwell International) разработали первую СУБД - иерархическую систему IMS (Information Management System). Несмотря на то, что IMS является самой первой из всех коммерческих СУБД, она до сих пор остается основной иерархической СУБД, используемой на большинстве крупных мейнфреймов.

Другим заметным достижением середины 60-х годов было появление системы IDS (Integrated Data Store) фирмы General Electric. Развитие этой системы привело к созданию нового типа систем управления базами данных -

сетевых СУБД, что оказало существенное влияние на информационные системы того поколения. Сетевая СУБД создавалась для представления более сложных взаимосвязей между данными, чем те, которые можно было моделировать с помощью иерархических структур, и послужили основой для разработки первых стандартов БД. Для создания таких стандартов в 1965 году на конференции CODASYL (Conference on Data Systems Languages) была сформирована рабочая группа List Processing Task Force, переименованная в 1967 году в группу Data Base Task Group (DBTG). В компетенцию группы DBTG входило определение спецификаций среды, которая допускала бы разработку баз данных и управление данными.

Полный вариант отчета этой группы был опубликован в 1971 году и содержал следующие утверждения:

- Сетевая схема - это логическая организация всей базы данных в целом (с точки зрения АДБ), которая включает определение имени базы данных, типа каждой записи и компонентов записей каждого типа.
- Подсхема - это часть базы данных, видимая конкретными пользователями или приложениями.
- Язык управления данными - инструмент для определения характеристик и структуры данных, а также для управления ими.

Группа DBTG также предложила стандартизировать три различных языка:

- Язык определения данных DDL для схемы, который позволит АБД описать ее.
- Язык определения данных (также DDL) для подсхемы, который позволит определять в приложениях те части базы данных, доступ к которым будет необходим.
- Язык манипулирования данными DML, предназначенный для управления данными.

Несмотря на то что этот отчет официально не был одобрен Национальным Институтом Стандартизации США (American National Standards Institute - ANSI), большое количество систем было разработано в полном соответствии с этими предложениями группы DBTG. Теперь они называются CODASYL-системами, или DBTG-системами. CODASYL-системы и системы на основе иерархических подходов представляют собой СУБД первого поколения.

Однако этим двум моделям присущи приведенные ниже недостатки.

- Даже для выполнения простых запросов с использованием переходов и доступом к определенным записям необходимо создавать достаточно сложные программы.
- Независимость от данных существует лишь в минимальной степени.
- Отсутствуют теоретические основы.

В 1970 году Э. Ф. Кодд, работавший в корпорации IBM, опубликовал статью о реляционной модели данных, позволявшей устранить недостатки прежних моделей. Вслед за этим появилось множество экспериментальных реляционных СУБД, а первые коммерческие продукты появились в конце 70-х - начале 80-х годов. Особенно следует отметить проект System R, разработанный в корпорации IBM в конце 70-х годов. Этот проект был задуман с целью доказать практичность реляционной модели, что достигалось посредством реализации предусмотренных ею структур данных и требуемых функциональных возможностей.

На основе этого проекта были получены важнейшие результаты:

- Был разработан структурированный язык запросов SQL, который с тех пор стал стандартным языком любых реляционных СУБД.
- В 80-х годах были созданы различные коммерческие реляционные СУБД - например, DB2 или SQL/DS корпорации IBM, Oracle корпорации Oracle , др.

В настоящее время существует несколько сотен различных реляционных СУБД для мейнфреймов и персональных ЭВМ. Реляционные СУБД относятся к

СУБД второго поколения. Однако реляционная модель также обладает некоторыми недостатками - в частности, ограниченными возможностями моделирования. Для решения этой проблемы был выполнен большой объем исследовательской работы. В 1976 году Чен предложил модель «сущность-связь» (Entity-Relationship model - ER-модель), которая в настоящее время стала основой методологии концептуального проектирования баз данных и методологии логического проектирования реляционных баз данных. В 1979 году Кодд сделал попытку устранить недостатки собственной основополагающей работы и опубликовал расширенную версию реляционной модели - RM/T (1979), затем еще одну версию - RM/V2 (1990). Попытки создания модели данных, позволяющей более точно описывать реальный мир, нестрого называют семантическим моделированием данных (semantic data modeling).

В ответ на все возрастающую сложность приложений баз данных появились две новые системы: объектно-ориентированные СУБД, или ОО СУБД (Object-Oriented DBMS - OODBMS), и объектно-реляционные СУБД, или ОР СУБД (Object-Relational DBMS - ORDBMS). Попытки реализации подобных моделей представляют собой СУБД третьего поколения.

2 Современные технологии обработки данных

Информационные технологии обработки данных предназначены для решения хорошо структурированных задач, по которым имеются необходимые входные данные и известны алгоритмы и другие стандартные процедуры их обработки. Эти технологии применяются на уровне исполнительской деятельности персонала невысокой квалификации в целях автоматизации некоторых рутинных постоянно повторяющихся операций управленческого труда. Внедрение информационных технологий и систем на этом уровне позволяет повысить производительность труда персонала, освободить его от рутинных операций, что, в конечном счете, позволяет сократить необходимое время обработки или численность работников.

2.1 Основные компоненты

Представим основные компоненты информационной технологии обработки данных и приведем их характеристики. Рассмотрим использования информационных технологий обработки данных на примере работы виртуальной фирмы занимающейся выпуском некоторой продукции из некоторого сырья.

На уровне операционной деятельности такой фирмы должны решаться следующие задачи:

- обработка данных об операциях, производимых фирмой;
- создание периодических контрольных отчетов о состоянии дел в фирме;
- получение ответов на всевозможные текущие запросы и оформление их в виде бумажных документов или отчетов.

Информационные технологии обработки данных могут использоваться, например, для автоматизации следующих рутинных операций:

- операция проверки на соответствие нормативу уровня запасов сырья на складе;

- операция учета расходов сырья на выпуск продукции;
- операция проверки наличия продукции на складе;
- операция продажи продукции фирмой.

При уменьшении уровня запаса система должна выдать заказ поставщику с указанием необходимого количества сырья и сроков поставки. При выпуске продукции система должна учитывать необходимое для этого количество сырья. Количество выпущенной продукции при этом не должно превышать некоторого максимального значения, определяемого возможностями склада готовой продукции. При продаже продукции система должна проверять ее наличие на складе и формировать выходной документ для покупателя в виде чека или квитанции.

Пример контрольного отчета: ежедневный отчет о наличии на складе сырья или готовой продукции, формируемый в целях контроля загруженности складов сырья и готовой продукции. Пример запроса: запрос к базе данных по складу сырья, который позволит получить данные о количестве и стоимости приобретенного сырья за определенный промежуток времени.

Можно выделить следующие основные компоненты информационной технологии рассматриваемой системе (рисунок 2.1):

- сбор данных;
- обработка данных;
- хранение данных;
- создание отчетов.

Сбор данных. По мере того как фирма производит продукцию, каждое ее действие должно сопровождаться соответствующими записями данных. Обычно действия фирмы, затрагивающие внешнее окружение, выделяются особо как операции, производимые фирмой.

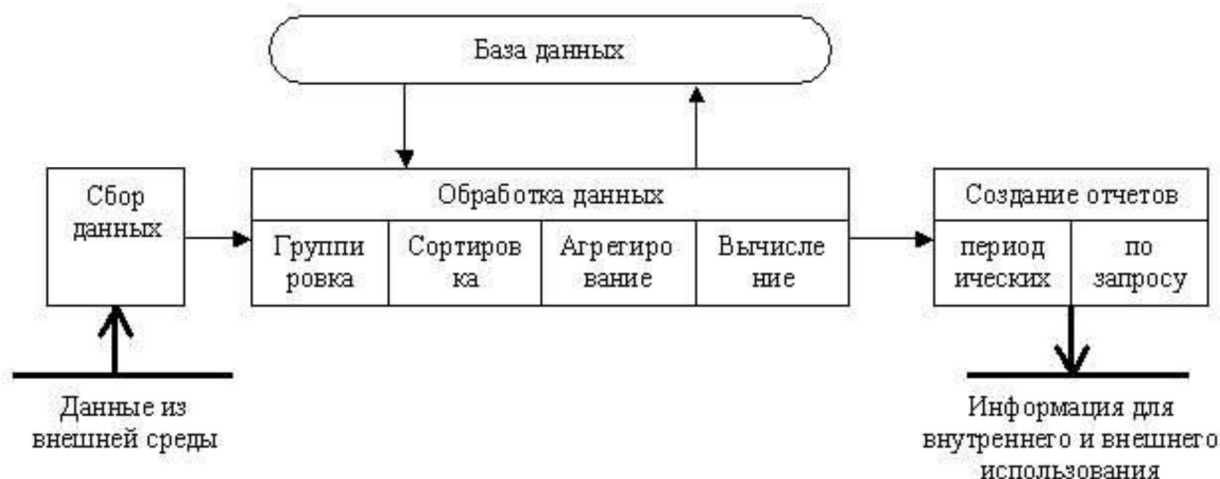


Рисунок 2.1 - Основные компоненты информационной технологии обработки данных

Обработка данных. Для создания из поступающих данных информации, отражающей деятельность фирмы, используются следующие типовые операции:

- группировка или классификация данных;
- сортировка или ранжирование данных;
- вычисления новых данных, включающие арифметические и логические операции;
- укрупнение или агрегирование информации.

Первичные данные обычно имеют вид кодов, состоящих из одного или нескольких символов. Эти коды, выражающие определенные признаки объектов, используются для идентификации и группировки записей. С помощью сортировки или ранжирования данные упорядочиваются по заданным признакам. Производные данные вычисляются на основе заданных исходных данных с использованием различного математического аппарата. Эти операции, выполняемые над данными, дают возможность получать новые данные. Укрупнение или агрегирование служит для уменьшения количества данных и реализуется в форме расчетов итоговых или средних значений.

Хранение данных. Многие данные на уровне операционной деятельности необходимо сохранять для последующего использования либо здесь же, либо на другом уровне. Для их хранения создаются базы данных.

Создание отчетов (документов). В информационной технологии обработки данных необходимо создавать документы для руководства и работников фирмы, а также для внешних партнеров. При этом документы или в связи с проведенной фирмой операцией так и периодически в конце каждого месяца, квартала или года.

Обработка данных является одним из главных компонентов большинства информационных систем. ИС в зависимости от специфики предметной области может работать в различных режимах и использовать различные способы обработки, на которых следует остановиться более подробно.

2.2 Режимы обработки данных

При проектировании технологических процессов обработки данных ориентируются на режимы их реализации. Режим реализации технологии зависит от объемно-временных особенностей решаемых задач: периодичности и срочности, требований к скорости обработки сообщений, а также от режимных возможностей технических средств, и в первую очередь ЭВМ.

Таким образом, можно выделить следующие режимы обработки данных:

- пакетный режим;
- режим реального масштаба времени;
- диалоговый.

Пакетный режим. При использовании этого режима пользователь не имеет непосредственного общения с ЭВМ. Сбор и регистрация информации, ввод и обработка не совпадают по времени. Вначале пользователь собирает информацию, формируя ее в пакеты в соответствии с видом задач или каким-то другим признаком. Как правило, это задачи неоперативного характера, с долгосрочным сроком действия результатов решения. После завершения

приема информации производится ее ввод и обработка, таким образом, происходит задержка обработки. Этот режим используется, как правило, при централизованном способе обработки информации.

Режим реального масштаба времени означает способность вычислительной системы взаимодействовать с контролируемыми или управляемыми процессами в темпе протекания этих процессов. Время реакции ЭВМ должно удовлетворять темпу контролируемого процесса или требованиям пользователей и иметь минимальную задержку. Как правило, этот режим используется при децентрализованной и распределенной обработке данных.

Диалоговый режим (запросный) режим, при котором существует возможность пользователя непосредственно взаимодействовать с вычислительной системой в процессе работы пользователя. Программы обработки данных находятся в памяти ЭВМ постоянно, если ЭВМ доступна в любое время, или в течение определенного промежутка времени, когда ЭВМ доступна пользователю. Взаимодействие пользователя с вычислительной системой в виде диалога может быть многоаспектным и определяться различными факторами: языком общения, активной или пассивной ролью пользователя; кто является инициатором диалога - пользователь или ЭВМ; временем ответа; структурой диалога и т.д. Если инициатором диалога является пользователь, то он должен обладать знаниями по работе с процедурами, форматами данных и т.п. Если инициатор - ЭВМ, то машина сама сообщает на каждом шаге, что нужно делать с разнообразными возможностями выбора. Этот метод работы называется "выбором меню". Он обеспечивает поддержку действий пользователя и предписывает их последовательность. При этом от пользователя требуется меньшая подготовленность.

Диалоговый режим требует определенного уровня технической оснащенности пользователя, т.е. наличие терминала или ПЭВМ, связанных с центральной вычислительной системой каналами связи. Этот режим

используется для доступа к информации, вычислительным или программным ресурсам. Возможность работы в диалоговом режиме может быть ограничена во времени начала и конца работы, а может быть и неограниченной.

2.3 Способы обработки данных

Способы обработки данных могут зависеть от различных факторов определяемых особенностями предметной области. Можно выделить следующие способы обработки данных:

- централизованная;
- децентрализованная;
- распределенная;
- интегрированная.

Централизованный способ обработки данных предполагает наличие вычислительного центра (ВЦ). При этом способе пользователь доставляет на ВЦ исходную информацию и получает результаты обработки в виде результативных документов. Особенностью такого способа обработки являются сложность и трудоемкость налаживания быстрой, бесперебойной связи, большая загруженность ВЦ информацией (т.к. велик ее объем), регламентацией сроков выполнения операций, организация безопасности системы от возможного несанкционированного доступа. Использование современных высокоскоростных компьютерных сетей значительно повышает эффективность централизованной обработки данных.

Децентрализованная обработка. Этот способ связан с появлением персональных компьютеров (ПК), дающих возможность автоматизировать конкретное рабочее место. В настоящее время существуют три вида технологий децентрализованной обработки данных.

Первая основывается на персональных компьютерах, не объединенных в локальную сеть (данные хранятся в отдельных файлах и на отдельных носителях информации). Недостатки данного вида: отсутствие связи между

задачами, невозможность обработки больших объемов информации, низкая защита от несанкционированного доступа.

Второй вид децентрализованной обработки данных заключается в объединении ПК в локальную сеть, что ведет к созданию единых файлов данных (но такой вид также не рассчитан на большие объемы информации).

Третий вид децентрализованной обработки данных на ПК, объединенных в локальную сеть, предполагает использование специальных серверов (такая архитектура называется «клиент-сервер»).

Распределенный способ обработки данных основан на распределении функций обработки между различными ЭВМ, включенными в сеть. Этот способ может быть реализован двумя путями: первый предполагает установку ЭВМ в каждом узле сети (или на каждом уровне системы), при этом обработка данных осуществляется одной или несколькими ЭВМ в зависимости от реальных возможностей системы и ее потребностей на текущий момент времени. Второй путь - размещение большого числа различных процессоров внутри одной системы.

Преимущества распределенного способа:

- возможность обрабатывать в заданные сроки любой объем данных;
- высокая степень надежности, так как при отказе одного технического средства есть возможность моментальной замены его на другой.;
- сокращение времени и затрат на передачу данных;
- повышение гибкости систем, упрощение разработки и эксплуатации программного обеспечения;
- и т.д.

Распределенный способ основывается на комплексе специализированных процессоров, т.е. каждая ЭВМ предназначена для решения определенных задач, или задач своего уровня.

Следующий способ обработки данных - *интегрированный*. Он предусматривает создание информационной модели управляемого объекта, то есть создание распределенной базы данных. Такой способ обеспечивает максимальное удобство для пользователя. С одной стороны, базы данных предусматривают коллективное пользование и централизованное управление. С другой стороны, объем информации, разнообразие решаемых задач требуют распределения базы данных. Технология интегрированной обработки информации позволяет улучшить качество, достоверность и скорость обработки, т.к. обработка производится на основе единого информационного массива, однократно введенного в ЭВМ. Особенностью этого способа является отделение технологически и по времени процедуры обработки данных от процедур сбора, подготовки и ввода данных.

Рассмотрим, каким образом соотносится распределенная обработка данных с распределенными базами данных более подробно.

3 Распределенная обработка данных и распределенные базы данных

Процессы децентрализации и информационной интеграции, происходящие во всем мире, неизбежно должны рано или поздно затронуть нашу страну. Россия, в силу своего географического положения и размеров "обречена" на преимущественное использование распределенных систем. Это направление может успешно развиваться лишь при выполнении двух главных условий - адекватном развитии глобальной сетевой инфраструктуры и применении реальных технологий создания распределенных информационных систем.

Важность этой темы осознают все. Действительно, страна прошла начальный этап локальной компьютеризации. Многие задачи "автоматизации в малом" или "автоматизации в среднем" уже решаются адекватными средствами на достаточно высоком технологическом уровне. Но вот задачи совершенно иного качества - задачи создания корпоративных информационных систем - нуждаются в осмыслении и анализе. Сложность нынешнего этапа во многом predetermined традиционализмом и инерционностью мышления, выражающейся в попытке переноса средств и решений локальной автоматизации в мир распределенных систем. Этот мир живет по своим законам, которые требуют иных технологий.

Как правило, компьютеры и программы, входящие в состав информационной системы, не являются равноправными. Некоторые из них владеют ресурсами (файловая система, процессор, принтер, база данных и т.д.), другие имеют возможность обращаться к этим ресурсам. Компьютер (или программу), управляющий ресурсом, называют сервером этого ресурса (файл-сервер, сервер базы данных, вычислительный сервер...). Клиент и сервер какого-либо ресурса могут находиться как в рамках одной вычислительной системы, так и на различных компьютерах, связанных сетью.

Основной принцип технологии "клиент-сервер" заключается в разделении функций приложения на три группы:

- ввод и отображение данных (взаимодействие с пользователем);
- прикладные функции, характерные для данной предметной области;
- функции управления ресурсами (файловой системой, базой данных и т.д.).

Поэтому, в любом приложении можно выделить следующие компоненты:

- компонент представления данных;
- прикладной компонент;
- компонент управления ресурсом.

Связь между компонентами осуществляется по определенным правилам, которые называют "протокол взаимодействия". Каждый из компонентов приложения при этом может работать на выделенном сервере (узле) или разделять ресурсы сервера с другими компонентами приложения. В связи с этим можно выделить следующие модели приложений:

- двухзвенная модель (модель «клиент-сервер»);
- трехзвенная модель (модель сервера приложений);
- многозвенная модель.

Двухзвенная модель позволяет распределить различным образом три компонента приложения между двумя узлами. Трехзвенная модель предполагает выделение для каждого из трех компонентов приложения свой сервер. Многозвенная модель позволяет отдельным компонентам использовать ресурсы нескольких серверов, например, распределенные базы данных.

Компанией Gartner Group, специализирующейся в области исследования информационных технологий, предложена следующая классификация двухзвенных моделей взаимодействия клиент-сервер (рисунок 3.1).

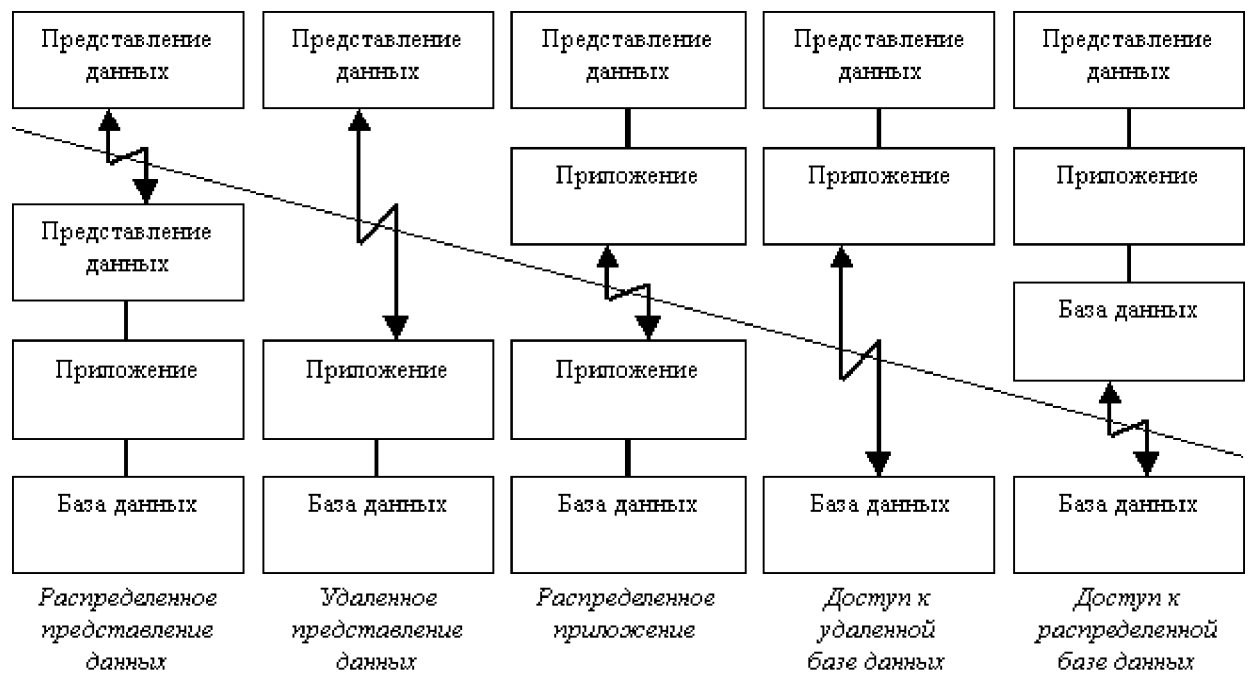


Рисунок - 3.1 Классификация двухзвенных моделей

Исторически первой появилась модель распределенного представления данных, которая реализовывалась на универсальной ЭВМ с подключенными к ней неинтеллектуальными терминалами. Управление данными и взаимодействие с пользователем при этом объединялись в одной программе, на терминал передавалась только "картинка", сформированная на центральном компьютере.

Затем, с появлением ПК и локальных сетей, были реализованы модели доступа к удаленной базе данных. Некоторое время базовой для сетей ПК была архитектура файлового сервера. При этом один из компьютеров является файловым сервером, на клиентах выполняются приложения, в которых совмещены компонент представления и прикладной компонент (СУБД и прикладная программа). Протокол обмена при этом представляет набор низкоуровневых вызовов операций файловой системы. Такая архитектура, реализуемая, как правило, с помощью персональных СУБД, имеет очевидные

недостатки - высокий сетевой трафик и отсутствие унифицированного доступа к ресурсам.

С появлением первых специализированных серверов баз данных появилась возможность другой реализации модели доступа к удаленной базе данных. В этом случае ядро СУБД функционирует на сервере, протокол обмена обеспечивается с помощью языка SQL. Такой подход по сравнению с файловым сервером ведет к уменьшению загрузки сети и унификации интерфейса "клиент-сервер". Однако сетевой трафик остается достаточно высоким, кроме того, по-прежнему невозможно удовлетворительное администрирование приложений, поскольку в одной программе совмещаются различные функции.

Позже была разработана концепция активного сервера, который использовал механизм хранимых процедур. Это позволило часть прикладного компонента перенести на сервер (модель распределенного приложения). Процедуры хранятся в словаре базы данных, разделяются между несколькими клиентами и выполняются на том же компьютере, что и SQL-сервер. Преимущества такого подхода: возможно централизованное администрирование прикладных функций, значительно снижается сетевой трафик (т.к. передаются не SQL-запросы, а вызовы хранимых процедур). Недостаток - ограниченность средств разработки хранимых процедур по сравнению с языками общего назначения (C и Pascal).

На практике часто используется смешанный подход:

- простейшие прикладные функции выполняются хранимыми процедурами на сервере;
- более сложные функции реализуются на клиенте непосредственно в прикладной программе.

В настоящее время получило широкое распространение использование модели распределенного приложения. Характерной чертой таких приложений является логическое разделение приложения на две и более частей, каждая из

которых может выполняться на отдельном сервере. Выделенные части приложения взаимодействуют друг с другом, обмениваясь сообщениями в заранее согласованном формате. В этом случае двухзвенная архитектура клиент-сервер становится трехзвенной (рисунок 3.2), а в некоторых случаях, она может включать и больше звеньев.



Рисунок 3.2 - Трехзвенная архитектура

С появлением современных высокоскоростных средств телекоммуникаций появилась возможность объединять удаленные серверы баз данных в единую систему. Таким образом, можно выделить два основных направления распределения приложений: системы распределенной обработки данных и системы распределенных баз данных.

3.1 Системы распределённой обработки данных

В современных сетевых информационных технологиях всё чаще используют распределённую обработку данных. Она позволяет повысить эффективность удовлетворения информационных потребностей пользователей, обеспечить гибкость и оперативность принимаемых им решений и др. Под распределённой обработкой данных понимают обработку приложений несколькими территориально разделенными ЭВМ.

Распределенная обработка данных (Distributed Data Processing, DDP) - это методика выполнения прикладных программ группой систем. При этом пользователь получает возможность работать с сетевыми службами и

прикладными процессами, расположенными в нескольких взаимосвязанных абонентских системах.

Распределённая обработка данных позволяет повысить эффективность удовлетворения информационных потребностей пользователей, обеспечивает гибкость и оперативность принимаемых ими решений.

Функции распределённой среды включают службы:

- каталогов, позволяющие клиентам находить серверы;
- удаленного вызова процедур;
- обслуживания файлов;
- безопасности данных;
- времени, синхронизирующие часы в абонентских системах.

Информационные системы, связанные с обработкой больших объемов информации, используют для хранения информации базы и банки данных. При этом в приложениях, связанных с обработкой данных, собственно управление базой данных может выполняться централизованно. 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*.

В общем случае режимы использования БД можно представить в виде дерева (рисунок 3.1.1) .

Системы распределенной обработки данных в основном связаны с первым поколением БД, которые строились на мультипрограммных операционных системах и использовали централизованное хранение БД на устройствах внешней памяти центральной ЭВМ и терминальный многопользовательский режим доступа к ней. При этом пользовательские терминалы не имели собственных ресурсов - то есть процессоров и памяти, которые могли бы использоваться для хранения и обработки данных.



Рисунок 3.1.1 - Режимы работы с базой данных

Первой полностью реляционной системой, работающей в многопользовательском режиме, была СУБД SYSTEM R, разработанная фирмой IBM, именно в ней были реализованы как язык манипулирования данными SQL, так и основные принципы синхронизации, применяемые при распределенной обработке данных, которые до сих пор являются базисными практически во всех коммерческих СУБД.

Общая тенденция движения от отдельных mainframe-систем к открытым распределенным системам, объединяющим компьютеры среднего класса, получила название DownSizing. Этот процесс оказал огромное влияние на развитие архитектур СУБД и поставил перед их разработчиками ряд сложных задач. Главная проблема состояла в технологической сложности перехода от централизованного управления данными на одном компьютере и СУБД, использовавшей собственные модели, форматы представления данных и языки доступа к данным и т. д., к распределенной обработке данных в неоднородной вычислительной среде, состоящей из соединенных в глобальную сеть компьютеров различных моделей и производителей.

В то же время происходил встречный процесс - UpSizing. Бурное развитие персональных компьютеров, появление локальных сетей также оказали

серьезное влияние на эволюцию СУБД. Высокие темпы роста производительности и функциональных возможностей РС привлекли внимание разработчиков профессиональных СУБД, что привело к их активному распространению на платформе настольных систем.

Сегодня возобладала тенденция создания информационных систем на такой платформе, которая точно соответствовала бы ее масштабам и задачам. Она получила название RightSizing (помещение ровно в тот размер, который необходим).

Однако и в настоящее время большие ЭВМ сохраняются и сосуществуют с современными открытыми системами. Причина этого проста - в свое время в аппаратное и программное обеспечение больших ЭВМ были вложены огромные средства: в результате многие продолжают их использовать, несмотря на морально устаревшую архитектуру. В то же время перенос данных и программ с больших ЭВМ на компьютеры нового поколения сам по себе представляет сложную техническую проблему и требует значительных затрат.

Одновременная работа пользователей с базой данной, распределенной по нескольким компьютерам, расположенным в сети, называется *параллельным доступом к распределенным БД*. Подобные системы называются *системами распределенных баз данных*.

3.2 Системы распределенных баз данных

Наиболее часто данные размещаются в БД. Ими обычно управляют локальные СУБД, то есть размещённые на том же компьютере. Когда несколько таких БД удалены друг от друга на большие расстояния, то возникает необходимость решения задач управления ими, то есть распределёнными БД. Для решения таких задач между ЭВМ с локальными СУБД и БД организуют сеть передачи данных по каналам связи, а в ней обеспечивают техническую и программную поддержку обмена данными. То есть в этом случае используют

ПО, управляющее распределёнными базами данных, которые могут образовывать банки данных.

Распределенные базы данных (Distributed Data Base, DDB) представляют определённым образом связанные между собой БД, рассредоточенные на какой-либо территории (локально или регионально), обеспечивающие свободный обмен информацией и поиск данных в них.

Распределённая база данных предполагает хранение и выполнение функций управления данными в нескольких узлах и передачу данных между этими узлами в процессе выполнения запросов. Разбиение данных в распределённой базе данных может достигаться путём хранения различных таблиц на разных компьютерах или даже хранения разных частей и фрагментов одной таблицы на разных компьютерах. Для пользователя или прикладной программы не имеет значения, каким образом распределены данные между компьютерами. Работа с распределённой базой данных осуществляется так же, как и с централизованной, т. е. размещение БД должно быть прозрачно.

При распределённой обработке работа с базой (представление данных, их обработка и др.) ведётся на компьютере клиента, а поддержание базы в актуальном состоянии – на сервере. При этом такие БД обычно располагаться на нескольких серверах – различных узлах компьютерной сети, а некоторые данные могут дублироваться.

Размещение частей общей БД бывает *избыточным* или *безызыточным*. При избыточном размещении определяют степень дублирования частей (фрагментов) единой БД. Чтобы поддерживать целостность БД необходимо постоянно корректировать все её копии. Преимущества дублирования уменьшаются, когда увеличивается стоимость хранения её частей, что связано с необходимостью обеспечивать устойчивость системы.

Создание распределённых баз данных (РБД) вызвано попыткой одновременного решения двух задач: интеграции и децентрализации.

Интеграция подразумевает централизованное управление и ведение баз данных. *Децентрализация* обеспечивает хранение данных там, где они появились и обрабатываются. При решении этих задач снижается стоимость системы и увеличивается степень её надёжности, а также повышается скорость обработки данных. Решение задач интеграции и децентрализации распределённых баз данных привело к появлению нового вида программного обеспечения, на котором остановимся более подробно.

3.3 Системы управления распределёнными базами данных

Доступ пользователей к распределённым базам данных и их администрирование осуществляются с помощью системы управления распределённой базой данных (СУРБД).

Система управления распределёнными базами данных (Distributed dataBase management system, DDBMS) - это система управления базами данных, расположенными в нескольких узлах информационной сети. В СУРБД используется комбинация централизованного и локального способов хранения данных.

Для решения задач с распределёнными БД, во-первых, необходимо организовать между этими ЭВМ сеть передачи данных, то есть соединить их каналами связи. Затем обеспечивают техническую и программную поддержку обмена данными между ними, образуя тем самым сеть ЭВМ.

СУРБД создаются таким образом, чтобы максимально обеспечить соблюдение принципа независимости прикладных программ от локализации данных в сети. При этом логическое представление распределённой БД и манипулирование данными для прикладной программы ничем не отличаются от работы пользователя с локальной базой. Такие СУРБД оснащены каталогами, в которых хранятся структура сети, информация о локальных СУРБД и базах данных, а также программным обеспечением, которое на основе этой

информации управляет взаимодействием прикладной программы и конкретной локальной базой данных сети.

Сложность управления распределёнными базами данных во многом зависит от того, поддерживаются ли они однотипными локальными СУРБД, взаимодействие между которыми осуществляется просто. В противном случае в такую сеть включают различные программные и технические устройства, обеспечивающие единый интерфейс, согласование и возможность выполнения информационных процессов, например, использовать промежуточную интерфейсную СУРБД и др.

Заключение по главе 3

Таким образом, технология распределённой обработки данных представляет стандартный набор сетевых служб для выполнения прикладных процессов, рассредоточенных по группе абонентских систем (по гетерогенной сети). При этом создаются распределённые базы и банки данных. Обработка информации в базе ведётся на компьютере клиента, а поддержание базы в актуальном состоянии – на сервере. Доступ пользователей к БД и администрирование ею осуществляются с помощью системы управления распределённой базой данных (СУРБД). Подобные системы дают возможность пользователям обращаться к любым хранящимся в них данным и порой предоставляют новые, ранее неизвестные, возможности работы с информацией. При этом возникают новые проблемы, решаемые путём использования новых технологий.

4 Модели распределенной обработки данных в современных СУБД

Распределенная обработка данных исходно связана с парадигмой открытых систем, которая появилась в 90-х годах и быстро эволюционировала. Ранее приложение (пользовательская программа) не разделялась на части, оно выполнялось некоторым монолитным блоком. Но возникла идея более рационального использования ресурсов сети. Действительно, при монолитном исполнении используются ресурсы только одного компьютера, а остальные компьютеры в сети рассматриваются как терминалы. Но теперь, в отличие от эпохи main-фреймов, все компьютеры в сети обладают собственными ресурсами, и разумно так распределить нагрузку на них, чтобы максимальным образом использовать их ресурсы.

Для распределения нагрузки между узлами сети необходимо разработать модель разбиения единого монолитного приложения на отдельные части и определить принципы взаимосвязи между этими частями. Рассмотрим типовое интерактивное приложение, работающее с системой управления базой данных.

Интерактивные приложения, работающие с базами данных, выполняют функции, которые можно объединить в следующие группы (рисунок 4.1):

- 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.

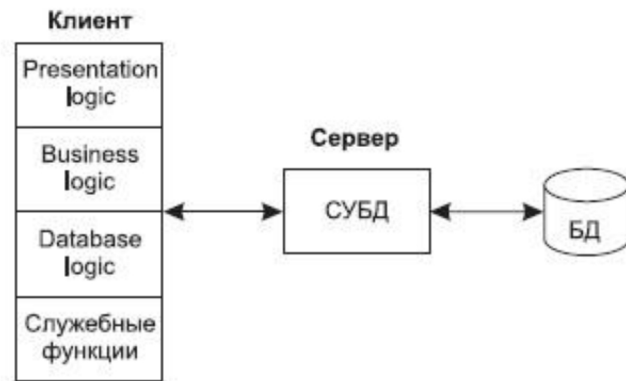


Рисунок 4.1 - Структура типового интерактивного приложения, работающего с базой данных

Презентационная логика (Presentation Logic) как часть приложения определяется тем, что пользователь видит на своем экране, когда работает приложение. 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.

Таким образом, основными задачами презентационной логики являются:

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

Некоторые возможности для организации презентационной логики приложений предоставляет знако-ориентированный пользовательский интерфейс, задаваемый моделями CICS (Customer Control Information System) и IMS/DC фирмы IBM и моделью TSO (Time Sharing Option) для централизованной main-фреймовой архитектуры. Модель GUI — графического пользовательского интерфейса, поддерживается в операционных средах Microsoft Windows, в OS/2 Presentation Manager, X-Windows и OSF/Motif.

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++, Cobol, SmallTalk, Visual-Basic.

Логика обработки данных (Data manipulation Logic) — это часть кода приложения, которая связана с обработкой данных внутри приложения. Данными управляет собственно СУБД (DBMS). Для обеспечения доступа к данным используются язык запросов и средства манипулирования данными стандартного языка SQL.

Обычно операторы языка SQL встраиваются в языки 3-го или 4-го поколения (3GL, 4GL), которые используются для написания кода приложения.

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 нам 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 (рисунок 4.2):

- 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).



Рисунок 4.2 - Распределение функций приложения

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.

Таким образом, в зависимости от распределения функций между различными компьютерами сети можно выделить следующие модели: двухзвенная, трехзвенная и модель сервера приложений.

4.1 Двухзвенная модель

Двухзвенная модель фактически является результатом распределения пяти указанных функций между двумя процессами, которые выполняются на двух платформах: на клиенте и на сервере.

Можно выделить следующие виды двухзвенной модели:

- модель удаленного управления данными (модель файлового сервера);
- модель удаленного доступа к данным;
- модель активного сервера баз данных.

В чистом виде почти никакая модель не существует, однако рассмотрим наиболее характерные особенности каждой двухуровневой модели.

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 (рисунок 4.3).



Рисунок 4.3 - Модель файлового сервера

В этой модели файлы базы данных хранятся на сервере, клиент обращается к серверу с файловыми командами, а механизм управления всеми информационными ресурсами, собственно база мета-данных, находится на клиенте.

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.

Рассмотрим алгоритм выполнения запроса клиента. Запрос клиента формулируется в командах языка манипулирования данными (ЯМД) выбранной СУБД. СУБД переводит этот запрос в последовательность файловых команд. Каждая файловая команда вызывает передачу блока информации на клиента, далее на клиенте СУБД анализирует полученную информацию, и если в полученном блоке не содержится ответ на запрос, то принимается решение о передаче следующего блока информации и т. д.

Передача информации с сервера на клиент производится до тех пор, пока не будет получен ответ на запрос клиента.

Недостатки модели файлового сервера:

- 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).

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 рисунке 4.4.

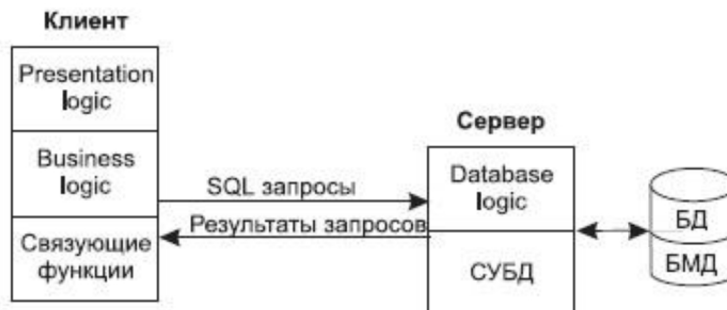


Рисунок 4.4 - Модель удаленного доступа

Преимущества данной модели:

- перенос компонента представления и прикладного компонента на клиентский компьютер существенно разгрузил сервер 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. (Это становится возможным, если отказаться от терминалов, не располагающих ресурсами, и заменить их компьютерами, выполняющими роль клиентских станций, которые обладают собственными локальными вычислительными ресурсами);
- 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 модели удаленного доступа:

- 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, например, 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. В случае, если объем остатка 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.

Для того чтобы избавиться от недостатков модели удаленного доступа, должны быть соблюдены следующие условия:

1. Необходимо, чтобы БД в каждый момент 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 языке описания данных (data definition language, DDL), который является частью SQL. Однако в реальных предметных областях у нас 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: Oracle, DB2, MS SQL Server. Основу данной модели составляет механизм хранимых процедур как средство программирования SQL-сервера, механизм триггеров как механизм отслеживания текущего состояния информационного хранилища и механизм ограничений на пользовательские типы данных, который иногда называется механизмом поддержки доменной структуры. Модель сервера баз данных представлена на рисунке 4.5.



Рисунок 4.5 - 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 регистрирует 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 (например, в Oracle язык PL-SQL, в MS SQL Server язык Transact 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 мы переложили на сервер большую 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".

Для разгрузки сервера в модели с активным сервером баз данных была предложена трехзвенная модель или модель сервера приложений.

4.2 Модель сервера приложений

Трехзвенная модель или модель сервера приложений является расширением двухзвенной модели и в ней вводится дополнительный промежуточный уровень между клиентом и сервером. Архитектура трехзвенной модели приведена на рисунке 4.6. Этот промежуточный уровень содержит один или несколько серверов приложений.

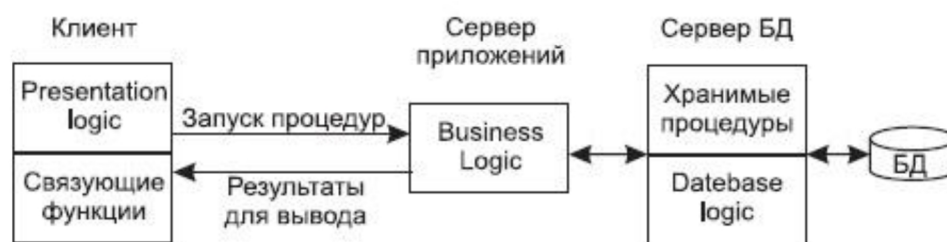


Рисунок 4.6 - Модель сервера приложений

В этой модели компоненты приложения делятся между тремя исполнителями:

- клиент;
- серверы приложений;
- серверы баз данных.

Клиент обеспечивает логику представления, включая графический пользовательский интерфейс, локальные редакторы; клиент может запускать локальный код приложения клиента, который может содержать обращения к

локальной БД, расположенной на компьютере-клиенте. Клиент выполняет коммуникационные функции front-end части приложения, которые обеспечивают доступ клиенту в локальную или глобальную сеть. Дополнительно реализация взаимодействия между клиентом и сервером может включать в себя управление распределенными транзакциями, что соответствует тем случаям, когда клиент также является клиентом менеджера распределенных транзакций.

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

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

Отметим, что эта модель обладает большей гибкостью, чем двухуровневые модели. Наиболее заметны преимущества модели сервера приложений в тех случаях, когда клиенты выполняют сложные аналитические расчеты над базой данных, которые относятся к области OLAP-приложений. (On-line analytical processing.) В этой модели большая часть бизнес-логики клиента изолирована от возможностей встроенного SQL, реализованного в конкретной СУБД, и может быть выполнена на стандартных языках

программирования, таких как C, C++, SmallTalk, Cobol. Это повышает переносимость системы, ее масштабируемость.

Функции промежуточных серверов могут быть в этой модели распределены в рамках глобальных транзакций путем поддержки XA-протокола (X/Open transaction interface protocol), который поддерживается большинством поставщиков СУБД.

Модель сервера приложений стала активно использоваться в глобальной сети Internet. В таком случае в сервер приложений тесно интегрируется с web-сервером, например, Apache. Такой подход зарекомендовал себя как один из методов публикации баз данных в Internet. Остановимся на этой проблеме более подробно.

4.3 Публикация баз данных в Internet

Появление Internet открыло новую эпоху в истории человечества. Internet недаром так быстро завоевал широкую популярность среди пользователей в мире бизнеса, науки, политики и т. д. Основные достижения Internet - это простота опубликования информации в сети, удобство и сравнительная унифицированность доступа к документам, наличие на сегодняшний день достаточно развитых средств поиска.

В настоящее время наиболее популярным средством доступа к информации в Internet является служба "Всемирной Паутины" (World Wide Web). Как известно, в ее основе лежит идея распределенной гипертекстовой информации. Явная навигация в Web позволяет добраться до самых разнообразных информационных серверов, содержащих специально подготовленную информацию (с использованием языка HTML).

Однако в целом способы представления, хранения и поиска информации в WWW относятся к категории информационно-поисковых систем (ИПС). Хотя хранилища данных в узлах Web иногда называют базами данных, этот термин в данном случае можно использовать только в самом широком смысле.

Исторически ИПС применялись для хранения слабоструктурированной и редко изменяемой информации. Базы данных в узком смысле - это хранилища структурированной, изменяемой информации, причем информация в базе данных должна всегда находиться в согласованном состоянии.

Базы данных содержат основные знания человечества. В конце двадцатого века с появлением технологии баз данных было накоплено больше информации, чем за всю предыдущую историю. В мире существует масса информационных источников, владельцы которых готовы предоставить их в пользование человечества, но не могут этого сделать по причинам недостатков технологии Internet (а иногда просто потому, что незнакомы с имеющимися технологиями).

Проблема заключается в том, что доступ к базам данных (даже к тем, которые содержат полностью открытую информацию) ограничен. Интерфейсы доступа к разным базам данных абсолютно различаются, так же, как и способы подключения баз данных к Internet. Имеются трудности как у тех, кто хочет пользоваться базами данных, так и у тех, кто хотел бы передать свою информацию в использование в режиме «on-line».

Чтобы получить интересующую информацию, пользователь должен иметь физический доступ к соответствующей СУБД, быть в курсе модели данных, знать схему базы данных и, наконец, уметь пользоваться соответствующим языком запросов. Что касается языка запросов, то проблему частично решает протокол ODBC, позволяющий направлять ограниченный набор операторов SQL (с промежуточной обработкой соответствующим драйвером ODBC) к произвольному серверу баз данных. Но это только частичное решение, поскольку оно никак не помогает пользователю понять схему базы данных (даже в терминах SQL) и, конечно, не способствует созданию унифицированного интерфейса конечного пользователя (нельзя же заставить всех работать в строчном режиме на языке SQL).

Кроме того, имеются удобные средства разработки распределенных в Internet гипертекстовых документов, простые, удобные, развитые и унифицированные интерфейсы для доступа к информации WWW, большое количество ценных баз данных, управляемых разнородными СУБД, а также желание сделать эти базы доступными всем людям (в случае публичных баз данных) или членам территориально-распределенной корпорации (в случае корпоративных баз данных). Возникает естественное желание скрестить эти две технологии и обеспечить доступ к базам данных в интерфейсе Web.

С точки зрения технологий баз данных идеальным решением было бы представление информации, накопленной в Internet, в виде одной распределенной структурированной базы данных в единой модели и с общей схемой. К сожалению, это невозможно по нескольким причинам:

- накоплен слишком большой объем неструктурированной информации, структуризация которой трудоемка и далеко не всегда возможна;
- содержимое Web-узлов разрабатывалось в расчете на использование гипертекстовых структур, поэтому даже если бы удалось структурировать отдельные документы, неизвестно, можно ли было бы описать хотя бы локальную схему узла (в смысле технологий баз данных);
- даже если бы удалось перевести все Web-узлы в структурированную форму, то пришлось бы решать неподъемную задачу интеграции несметного количества разнородных баз данных (напомню, что это очень сложно даже при небольшом числе интегрируемых баз данных);
- а самое главное, что для подавляющего большинства пользователей Internet (которые смотрят фильмы, слушают музыку, посещают виртуальные магазины и т. д., то есть используют Internet для удовлетворения своих бытовых потребностей) вся эта суэта была бы совершенно непонятна; их вполне устаивает нынешнее положение дел (если не считать упоминавшихся неприятностей, связанных с поиском нужной информации).

С другой стороны, Internet-сообщество в целом в полной мере осознало недостатки неструктурированных данных. Простота публикации не уравнивает сложность и ограниченность средств поиска.

Одним словом, сложилась ситуация, когда специалисты двух областей (Web и базы данных) одновременно неудовлетворенны сложившейся в Internet ситуацией, но не могут осуществить каких-либо решительных изменений. Необходим компромисс между простотой и доступностью применяемых в настоящее время Web-технологий и сложностью, функциональностью и эффективностью технологий баз данных.

Большинство компаний, производящих серверы баз данных и средства проектирования и разработки информационных приложений, в настоящее время озабочены проблемой обеспечения доступа к базам данных в среде Web. Имеются некоторые основания надеяться, что такой компромисс удастся выработать на основе языка разметки XML (eXtensible Markup Language).

На сегодняшний момент имеется несколько работающих механизмов. В целом механизмы делятся на два класса: обеспечивающие доступ к базе данных (по запросу клиента) на стороне Web-сервера и работающие непосредственно на стороне клиента.

Рассмотрим механизм доступ к базе данных на стороне сервера. Механизм реализуется за счет наличия двух более или менее стандартизованных средств: возможности включения форм в документ, составленный с использованием языка гипертекстовой разметки HTML, и возможности использования внешних по отношению к серверу Web программ, взаимодействие которых происходит через специфицированный протокол CGI (Common Gateway Interface) или внедренный позже API (Application Program Interface). Хотя CGI называется «общим интерфейсом шлюзования», по сути дела, это одновременно некоторое подмножество протокола HTTP, и способ его соблюдения при взаимодействии сервера с внешней программой.

Модель распределенной обработки данных в рамках Web-технологий, в случае реализации доступа к базе данных на стороне сервера, можно представить так (рисунок 4.7):

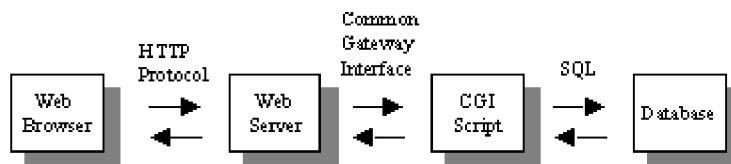


Рисунок 4.7 - Модель распределенной обработки данных в рамках web-технологий

Из рисунка видно, что в состав любого приложения Web баз данных входит компонент, который реализуют CGI скриптом. Однако это не единственный компонент таких приложений. Кроме него в состав приложения входит компонент отображения результатов обработки базы данных. Его часто реализуют на языке HTML.

Таким образом, Web приложения баз данных (WebDBApp) - это набор компонент, который в общем случае можно графически представить так (рисунок 4.8):

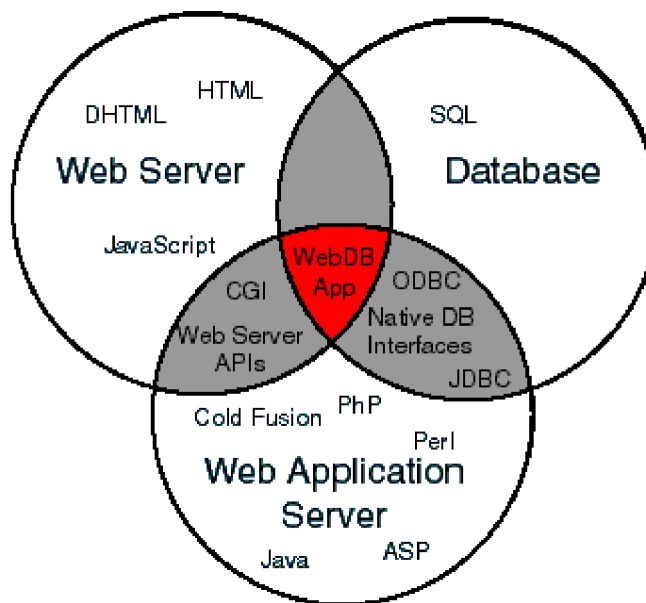


Рисунок 4.8 - Набор компонентов Web приложений баз данных

Как видно WebDBApp можно создавать на основании различных комбинаций компонент. Например, достаточно распространенная на сегодняшний день схема публикации баз данных в Internet: СУБД+язык PHP+WebServer. В данной схеме язык PHP реализует протокол CGI и позволяет получить доступ к базам данных.

При реализации на основе CGI общая схема реализации доступа к базе данных на стороне Web-сервера выглядит следующим образом:

- при просмотре документа клиент встречает ссылку на страницу, содержащую одну или несколько форм, предназначенных для запроса данных из базы данных;
- клиент запрашивает эту страницу; помимо незаполненных форм страница может содержать общую информацию о базе данных и о назначении предлагаемых форм;
- если клиента действительно интересует информация из базы данных, которую можно получить на основе предложенных форм, то он заполняет одну из форм и отправляет заполненную форму на сервер;
- получив заполненную форму, сервер запускает соответствующую внешнюю программу, передавая ей параметры и получая результаты на основе протокола CGI;
- внешняя программа преобразует запрос, выраженный с помощью заполненной формы, в запрос на языке, понятном серверу баз данных (обычно это язык SQL).
- внешняя программа взаимодействует с сервером баз данных; взаимодействие может быть прямым, если внешняя программа жестко привязана к конкретному SQL-серверу, или с использованием, например, протокола и соответствующего драйвера ODBC, если жесткая привязка отсутствует;

- после получения результатов запроса внешняя программа формирует соответствующую виртуальную или реальную HTML-страницу, передает ее серверу и завершает свое выполнение;
- сервер передает сформированную HTML-страницу клиенту, и на этом процедура доступа к базе данных завершается (как обычно, сервер разрывает транспортное соединение с клиентом).

Внешняя программа осуществляет компиляцию языка форм в базовый язык сервера баз данных (например, язык структурированных запросов SQL). Внешняя программа, запускаемая Web-сервером в соответствии со спецификациями CGI, называется CGI-скриптом. CGI-скрипт может быть написан на языке программирования (Си, Си++, Паскаль и т. д.), на командном языке (языки семейства shell, perl и т. д.) или на процедурном расширении базового языка сервера баз данных (например, PL-SQL, Transact-SQL и т.д.). CGI-скрипт, выполняющий роль посредника между Web-сервером и другими видами серверов (например, сервером баз данных), называется шлюзом (видимо, более правильно было бы использовать термин CGI-шлюз).

Наличие CGI-скриптов на стороне Web-сервера позволяет, в частности, перенести часть логики приложения из клиента в сервер. CGI-шлюзы представляют собой средство для организации трехзвенной (в общем случае, многозвенной) архитектуры клиент-сервер. В спецификациях CGI предусмотрены несколько способов взаимодействия Web-сервера и CGI-скрипта.

Первый способ состоит в использовании создаваемых сервером переменных окружения, через которые передается как общая информация, независимая от функциональных особенностей CGI-скрипта (например, имя и версия Web-сервера), так и специфические данные, определяющие поведение CGI-скрипта (скажем набор значений, введенных в форму на стороне клиента).

Второй способ заключается в формировании параметров `argc` и `argv`, которые передаются функции `main` CGI-скрипта (как если бы CGI-скрипт вызывался командной строкой в интерактивном режиме). Этот способ применяется для реализации ограниченного класса запросов ISINDEX. Наконец, входные параметры могут передаваться CGI-скрипту через файл стандартного ввода, а CGI-скрипт может передавать Web-серверу результирующие данные через файл стандартного вывода.

Как видно, при использовании CGI вся интерпретация пользовательского запроса производится CGI-скриптом. Скрипт может быть предельно жестким, ориентированным на выполнение запроса к фиксированной таблице фиксированной базы данных, или относительно гибким, способным выполнить произвольный запрос к одной или нескольким таблицам базы данных, идентифицируемой в параметрах клиента.

Запуск независимого "тяжеловесного" процесса стоит немало. Загрузка и выполнение еще одной программы в уже существующем адресном пространстве по сравнению с предыдущим вариантом гораздо дешевле.

API - это, фактически, дешевый способ (хотя и небезопасный) выполнить в адресном пространстве сервера WWW программу, которая соответствует спецификациям на языке HTML. Такая программа должна быть заранее подготовлена и включена в библиотеку, из которой сервер может производить динамическую загрузку (например, такая библиотека может быть разделяемой - `shared library`).

Теперь рассмотрим второй класс: механизм доступ к базе данных на стороне клиента. Видимо, наиболее мощные средства обеспечения доступа к базам данных на стороне Web-клиента обеспечивает язык Java. Java - это объектно-ориентированный язык программирования, являющийся, по сути дела, "безопасным" подмножеством языка Си++. В частности, Java не содержит средств адресной арифметики, не поддерживает механизм множественного

наследования и т. д. Поэтому утверждается, что корректность Java-программы можно проверить до ее реального выполнения (это абсолютно недоказанное утверждение). Различают:

- язык Java, как таковой, для которого существуют компиляторы в так называемый "мобильный код" (машинно-независимый код, который может интерпретироваться или из которого могут генерироваться машинные коды на разных платформах);
- язык JavaScript, который обычно используется для расширения возможностей языка HTML за счет добавления процедурной составляющей;
- и программный продукт HotJava, являющийся, по сути, интерпретатором мобильных кодов Java.

Для обеспечения доступа к базам данных на стороне Web-клиента наиболее существенно наличие языка Java. Технология разработки HTML-документа позволяет написать произвольное количество дополнительных Java-программ, откомпилировать их в мобильные коды и поставить ссылки на соответствующие коды в теле HTML-документа. Такие дополнительные Java-программы называются апплетами (Java-applets). Получив доступ к документу, содержащему ссылки на апплеты, клиентская программа просмотра запрашивает у Web-сервера все мобильные коды. Коды могут начать выполняться сразу после размещения в компьютере клиента или быть активизированы с помощью специальных команд.

Поскольку апплет представляет собой произвольную Java-программу, то, в частности, он может быть специализирован для работы с внешними базами данных. Более того, система программирования Java включает развитый набор классов, предназначенных для поддержки графического пользовательского интерфейса. Опираясь на использование этих классов, апплет может получить от пользователя информацию, характеризующую его запрос к базе данных, в

том же виде, как если бы использовался стандартный механизм форм языка HTML, а может применять какой-либо другой интерфейс.

Для взаимодействия Java-апплета с внешним сервером баз данных разработан специализированный протокол JDBC, который, фактически, сочетает функции шлюзования между интерпретатором мобильных Java-кодов и ODBC, а также включает ODBC.

В заключение сравним достоинства и недостатки двух рассмотренных подходов. Использование CGI-скриптов на стороне Web-сервера позволяет иметь на стороне клиента только сравнительно простые программы просмотра. Вся бизнес-логика работы с базами данных (возможно, с обработкой полученных данных) переходит на сторону Web-сервера. Это легкий способ построения трехзвенной архитектуры приложения. В последнее время разгрузку клиента от логики приложения называют решением проблемы "толстого" клиента. В данном случае мы можем иметь предельно тонкого клиента при утолщении сервера. Отрицательным моментом является то, что при необходимости подключения нового CGI-скрипта, вообще говоря, требуется (относительная) модификация кода сервера.

Использование Java-апплетов, вообще говоря, обеспечивает более гибкое решение. Апплет - это часть HTML-документа. Для включения нового апплета нужно всего-навсего перекомпоновать документ, при этом, Web-сервер изменять не нужно. С другой стороны, клиент должен быть толще.

4.4 Обеспечение безопасности данных

Система защиты информации - совокупность специальных мер правового и административного характера, организационных мероприятий, физических и технических средств защиты, а также специального персонала, предназначенного для обеспечения безопасности информации.

Правовые меры защиты информации - действующие в стране законы, указы и другие нормативные акты, регламентирующие правила обращения с информацией и ответственность за их нарушения.

Морально-этические меры защиты информации - традиционно сложившиеся в стране нормы поведения и правила обращения с информацией. Эти нормы не являются обязательными, как законодательно утвержденные нормы, однако, их несоблюдение ведет к падению авторитета, престижа человека, организации.

Организационные (административные) меры защиты - это меры, регламентирующие процессы функционирования АСОЭИ, использование ее ресурсов, деятельности персонала, а также порядок взаимодействия пользователей системой таким образом, чтобы максимально затруднить или исключить возможность реализации угроз безопасности информации.

Физические меры защиты - различные механические, электро- или электронно-механические устройства, предназначение для создания физических препятствий на путях проникновения потенциальных нарушителей к абонентам АБС и защищаемой информации, а также техник визуального наблюдения, связи и охранной сигнализации.

Технические (аппаратно-программные) средства защиты - различные электронные устройства и специальные программы, выполняющие (самостоятельно или в комплексе с другими средствами) функции защиты информации (идентификацию пользователей, разграничение доступа к ресурсам, криптографическое закрытие информации и т.п.)

Администратор безопасности - лицо или группа лиц, ответственных за обеспечение безопасности системы, за реализацию и непрерывность соблюдения установленных административных мер защиты и осуществляющих постоянную организационную поддержку функционирования применяемых физических и технических средств защиты.

Наилучшие результаты по защите информации достигаются при системном подходе к вопросам безопасности и комплексном использовании различных мер защиты на всех этапах жизненного цикла системы начиная с ее проектирования.

Существуют следующие универсальные (общие) способы защиты информации от различных воздействий на нее:

1. идентификация и аутентификация (пользователей процессов и т.д.);
2. контроль доступа к ресурсам системы (управление доступом);
3. регистрация и анализ событий, происходящих в системе;
4. контроль целостности объектов системы;
5. шифрование данных;
6. резервирование ресурсов, компонентов и данных системы.

Идентификация - это присвоение кода каждому объекту персонального идентификатора. Аутентификация - установление подлинности.

Управление доступом - защита информации путем регулирования доступа ко всем ресурсам системы (техническим, программным, элементам баз данных). Регламентируются порядок работы пользователей и персонала, право доступа к отдельным файлам в базах данных и т.д.

Резервирование ресурсов и компонентов информационной системы предполагает: организацию регулярных процедур спасения и резервного хранения критичных данных, установку и периодическую проверку резервных устройств обработки данных, подготовку специалистов, способных заменить администраторов систем, регистрацию систем и хранение носителей информации в строго определенных местах, выдачу их уполномоченным лицам с необходимыми отметками в регистрационных документах.

Важным компонентом в системе безопасности сетей передачи данных является цифровая подпись, позволяющая обмениваться юридически

значимыми документами, проводить платежные операции, подтверждать целостность передаваемой и проверять целостность полученной информации.

Современные системы управления базами данных имеют развитые механизмы обеспечения безопасности данных. Рассмотрим некоторые из таких механизмов.

В общем случае база данных является слишком дорогостоящим предметом, чтобы можно было использовать ее в автономном режиме. Обычно с достаточно большой базой данных (параллельно или последовательно) работает много приложений и пользователей, и не для всех них было бы разумно обеспечивать равноправный доступ к хранящимся данным.

В языке SQL (SQL:1999) предусмотрены возможности контроля доступа к разным объектам базы данных, в том числе к следующим объектам:

- таблицам;
- столбцам таблиц;
- представлениям;
- доменам;
- наборам символов;
- порядкам сортировки символов (collation);
- преобразованиям (translation);
- триггерам;
- подпрограммам, вызываемым из SQL;
- определенным пользователями типам.

В совокупности в SQL:1999 может поддерживаться девять видов защиты разных объектов в соответствии со следующими возможными действиями (Таблица 4.1).

Таблица 4.1 – Виды защиты

Вид защиты и соответствующее действие	Название привилегии	Применимо к следующим объектам
Просмотр	SELECT	Таблицы, столбцы, подпрограммы, вызываемые из SQL
Вставка	INSERT	Таблицы, столбцы
Модификация	UPDATE	Таблицы, столбцы
Удаление	DELETE	Таблицы
Ссылка	REFERENCES	Таблицы, столбцы
Использование	USAGE	Домены, определенные пользователями типы, наборы символов, порядки сортировки символов, преобразования
Инициирование	TRIGGER	Таблицы
Выполнение	EXECUTE	Подпрограммы, вызываемые из SQL
Подтипизация	UNDER	Структурные типы

При разработке средств контроля доступа к объектам баз данных создатели SQL придерживались принципа сокрытия информации об объектах, содержащихся в схеме базы данных, от пользователей, которые лишены доступа к этим объектам. Другими словами, если некоторый пользователь не обладает, например, привилегией на просмотр таблицы PRO, то при выполнении операции `SELECT * FROM PRO` он получит такое же диагностическое сообщение, как если бы таблица PRO не существовала. Если бы в случае отсутствия этой таблицы и в случае отсутствия привилегии доступа выдавались разные диагностические сообщения, то непривилегированный пользователь получил бы данные о том, что интересующая его таблица существует, но он лишен доступа к ней.

Привилегии доступа к объектам базы данных могут предоставляться пользователям, представляемым своими идентификаторами, а также ролям, выполнение которых, в свою очередь, может предоставляться пользователям. Кроме того, в SQL поддерживается концепция псевдоидентификатора (или идентификатора псевдо) пользователя PUBLIC, который соответствует любому

приложению или пользователю, зарегистрированному в системе баз данных. «Пользователю» PUBLIC могут предоставляться привилегии доступа к объектам базы данных, как и любому другому пользователю.

В модели контроля доступа SQL создатель любого объекта базы данных автоматически становится владельцем этого объекта. При этом владелец объекта может идентифицироваться либо своим идентификатором пользователя, либо именем своей роли. Вообще говоря, владелец объекта обладает полным набором привилегий для выполнения действий над объектом.

Владелец объекта, помимо прочего, обладает привилегией на передачу всех (или части) своих привилегий другим пользователям или ролям. В частности, владелец объекта может передать другим пользователям или ролям привилегию на передачу привилегий последующим пользователям или ролям (эти действия с передачей привилегии на передачу привилегий могут продолжаться рекурсивно).

Во многих реализациях поддерживаются привилегии уровня DBA (DataBase Administrator) для возможности выполнения операций DDL - Data Definition Language (CREATE, ALTER и DROP над объектами, входящими в схему базы данных). В стандарте SQL требуется лишь соблюдение следующих правил.

- Любые пользователь или его роль могут выполнять любые операции DDL внутри схемы, которой владеют.
- Не допускается выполнение каких-либо операций DDL внутри схемы, которой не владеет пользователь или роль, пытающиеся выполнить соответствующую операцию.
- Эти правила не допускают исключений.

В стандарте языка SQL не специфицированы средства создания идентификаторов авторизации. Если говорить более точно, в стандарте не определяется какой-либо явный способ создания допустимых идентификаторов

пользователей. Идентификатор авторизации может являться либо идентификатором пользователя, либо именем роли, а для создания ролей в SQL поддерживаются соответствующие средства. Но в соответствии с правилами стандарта SQL, все authID должны отслеживаться СУБД (имеются в виду все authID, для которых существует хотя бы одна привилегия). И в стандарте поддерживаются точные правила порождения и распространения привилегий. Привилегии по отношению к объекту базы данных предоставляются системой владельцу схемы при создании объекта в этой схеме, и привилегии могут явно передаваться от имени одного authID другому authID при наличии у первого authID привилегии на передачу привилегий.

Итак, authID может являться либо идентификатором пользователя, либо идентификатором роли. Попробуем разобраться в сути термина «роль». При работе с большими базами данных в крупных организациях часто сотни служащих производят над базой данных одни и те же операции. Конечно, для этого каждый из служащих должен быть зарегистрированным пользователем соответствующей системы баз данных и тем самым, обладать собственным authID. Используя базовые средства авторизации доступа (зафиксированные в стандарте SQL/92), можно предоставить каждому пользователю группы одни и те же привилегии доступа к требуемым объектам базы данных. Но схема авторизации доступа при этом становится очень сложной. В некотором смысле имя роли идентифицирует динамически образуемую группу пользователей системы баз данных, каждый из которых обладает, во-первых, привилегией на исполнение данной роли и, во-вторых, всеми привилегиями данной роли для доступа к объектам базы данных. Другими словами, наличие ролей упрощает построение и администрирование системы авторизации доступа. Проиллюстрируем это на рисунке 4.9.

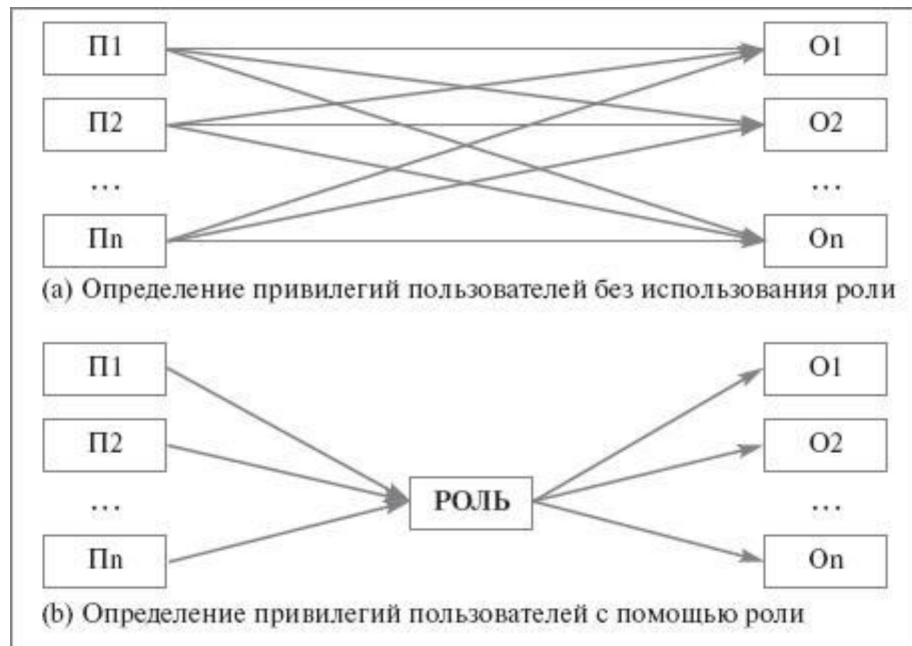


Рисунок 4.9 - Привилегии, пользователи и роли

Каждая стрелка на рисунке 4.9 соответствует мандату доступа (паре $\langle \text{authID}, \text{набор_привилегий_доступа_к_объекту_БД} \rangle$), который требуется сохранять в каталоге базы данных и проверять при попытке доступа от имени authID. Как видно, в случае (a) требуется сохранение и проверка $n \cdot m$ мандатов, где n – число пользователей в группе, а m – число объектов базы данных, для которых пользователи группы должны иметь одни и те же привилегии. В случае (b) число требуемых для корректной работы мандатов равно лишь $n + m$, и схема авторизации резко упрощается.

Группы пользователей, объединенных одной ролью, являются динамическими, поскольку в SQL поддерживаются возможности предоставления пользователю привилегии на исполнение данной роли и лишения пользователя этой привилегии. Более того, имеются возможности предоставления заданной роли 1 всех или части привилегий другой роли 1. Естественно, что при этом привилегии изменяются у всех пользователей, которые могут исполнять роль 1.

Более того, имеются возможности предоставления заданной роли А всех или части привилегий другой роли В. Естественно, что при этом привилегии изменяются у всех пользователей, которые могут исполнять роль А.

Для передачи привилегий и ролей от одних authID другим поддерживается оператор GRANT. Поскольку authID может являться идентификатором пользователя или именем роли, привилегии могут передаваться от пользователей пользователям, от пользователей ролям, от ролей ролям и от ролей пользователям. Один оператор GRANT позволяет передавать привилегии доступа только к одному объекту, но в том случае, когда объектом является таблица, разные привилегии могут передаваться по отношению к одному и тому же набору столбцов или к разным наборам.

Получателям передаваемых привилегий может быть выдана привилегия на дальнейшую передачу полученных привилегий, включая привилегию на передачу привилегий.

При проверке возможности выполнения операции в SQL-сессии учитываются привилегии текущего authID SQL-сессии, а также привилегии всех ролей, которые переданы данному authID. Поскольку этим ролям могли быть переданы другие роли, обладающие собственными привилегиями, анализ возможности выполнения операции является рекурсивной процедурой.

При определении представлений действуют специальные правила определения привилегий над этими представлениями. Если при создании обычных объектов базы данных, таких, как таблица или домен, текущий authID автоматически получает все возможные привилегии доступа к соответствующему объекту, включая привилегию на передачу привилегий, то для представлений ситуация иная. Поскольку создаваемое представление всегда основывается на одной или нескольких базовых таблицах (или представлениях), привилегии, которые получает создатель представления, должны основываться

на привилегиях, которыми располагает текущий authID по отношению к этим базовым таблицам или представлениям.

Например, чтобы операция создания представления была выполнена успешно, текущий authID должен обладать привилегией SELECT по отношению ко всем базовым таблицам и представлениям, на которых основывается новое представление. Тогда текущий authID автоматически получит привилегию SELECT для нового представления. Но текущий authID сможет передавать эту привилегию другим authID только тогда, когда обладает соответствующей привилегией для всех базовых таблиц и представлений, на которых основывается новое представление. Аналогичным образом на представление распространяются привилегии DELETE, INSERT, UPDATE и REFERENCES. Поскольку триггеры над представлениями создавать не разрешается, привилегия TRIGGER представлениям не передается.

Если от имени некоторого authID некоторые привилегии или роли были переданы одному или нескольким другим authID, то впоследствии первый authID (в сессии, где этот authID является текущим) можно изъять, или аннулировать, переданные привилегии или роли путем применения оператора REVOKE. Как и в случае передачи привилегий и ролей, способы аннулирования привилегий и ролей похожи, но между ними имеются некоторые отличия. Поэтому мы снова обсудим эти способы в отдельности.

Если некоторая привилегия была передана псевдо-authID PUBLIC, то, конечно, этой привилегией обладают все authID. Но нет возможности аннулировать такую привилегию у отдельно указываемого authID. Привилегия была передана всем, и аннулировать ее можно только сразу у всех.

5 Модели распределенных баз данных

Общая тенденция развития систем управления базами данных показывает, что распределенные системы получают все большее развитие и распространение. Этому способствует как мировая глобализация, которая приводит к ускорению процессов централизации и децентрализации информационных систем, так и технический прогресс в области хранения и передачи данных. Растут объемы и быстродействие накопителей информации. Все большей становится доля оптоволоконных каналов связи, которые позволяют передавать огромные массивы данных с невиданной до сих пор скоростью. Кроме того, ширится выпуск персональных домашних компьютеров и ноутбуков. Все это вкупе с прогрессирующим снижением стоимости коммуникационных услуг ведет к росту количества удаленных рабочих групп (офисов), а также расширению числа работников, выполняющих свои функции вне своего офиса или основного рабочего места. Эти факторы и способствуют развитию распределенных систем и, соответственно, распределенных баз данных.

Под распределенной базой обычно подразумевают базу данных, включающую фрагменты из нескольких баз данных, которые располагаются на различных узлах сети компьютеров, и, возможно, управляются различными СУБД. Распределенная база данных выглядит с точки зрения пользователей и прикладных программ как обычная локальная база данных. В этом смысле слово "распределенная" отражает способ организации базы данных, но не ее внешнюю характеристику.

Согласно принципам, изложенным в трудах известного ученого Дэйта (С. J. Date), можно выделить 12 основных требований к распределенной базе данных (они же являются основными признаками):

- локальная автономия (local autonomy);
- независимость узлов (no reliance on central site);

- непрерывные операции (continuous operation);
- прозрачность расположения (location independence);
- прозрачная фрагментация (fragmentation independence);
- прозрачное тиражирование (replication independence);
- обработка распределенных запросов (distributed query processing);
- обработка распределенных транзакций (distributed transaction processing);
- независимость от оборудования (hardware independence);
- независимость от операционных систем (operation system independence);
- прозрачность сети (network independence);
- независимость от баз данных (database independence).

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

Независимость от центрального узла. В идеальной системе все узлы равноправны и независимы, а расположенные на них базы являются равноправными поставщиками данных в общее пространство данных. База данных на каждом из узлов самодостаточна - она включает полный собственный словарь данных и полностью защищена от несанкционированного доступа.

Непрерывные операции. Это качество можно трактовать как возможность непрерывного доступа к данным (известное "24 часа в сутки, семь дней в неделю") в рамках DDB вне зависимости от их расположения и вне зависимости от операций, выполняемых на локальных узлах. Это качество можно выразить лозунгом "данные доступны всегда, а операции над ними выполняются непрерывно".

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

Прозрачная фрагментация. Это свойство трактуется как возможность распределенного (то есть на различных узлах) размещения данных, логически представляющих собой единое целое. Существует фрагментация двух типов: горизонтальная и вертикальная. Первая означает хранение строк одной таблицы на различных узлах (фактически, хранение строк одной логической таблицы в нескольких идентичных физических таблицах на различных узлах). Вторая означает распределение столбцов логической таблицы по нескольким узлам.

Рассмотрим пример, иллюстрирующий оба типа фрагментации. Имеется таблица employee (emp_id, emp_name, phone), определенная в базе данных на узле в Фениксе. Имеется точно такая же таблица, определенная в базе данных на узле в Денвере. Обе таблицы хранят информацию о сотрудниках компании. Кроме того, в базе данных на узле в Далласе определена таблица emp_salary (emp_id, salary). Тогда запрос "получить информацию о сотрудниках компании" может быть сформулирован так:

```
SELECT *  
FROM employee@phoenix, employee@denver  
ORDER BY emp_id
```

В то же время запрос "получить информацию о заработной плате сотрудников компании" будет выглядеть следующим образом:

```
SELECT employee.emp_id, emp_name, salary  
FROM employee@denver, employee@phoenix, emp_salary@dallas  
ORDER BY emp_id
```

Прозрачность тиражирования. Тиражирование данных - это асинхронный (в общем случае) процесс переноса изменений объектов исходной базы данных в базы, расположенные на других узлах распределенной системы. В данном контексте прозрачность тиражирования означает возможность переноса изменений между базами данных средствами, невидимыми пользователю распределенной системы. Данное свойство означает, что тиражирование возможно и достигается внутрисистемными средствами.

Обработка распределенных запросов. Это свойство DDB трактуется как возможность выполнения операций выборки над распределенной базой данных, сформулированных в рамках обычного запроса на языке SQL. То есть операцию выборки из DDB можно сформулировать с помощью тех же языковых средств, что и операцию над локальной базой данных. Например,

```
SELECT customer.name, customer.address, order.number, order.date  
FROM customer@london, order@paris  
WHERE customer.cust_number = order.cust_number
```

Обработка распределенных транзакций. Это качество DDB можно трактовать как возможность выполнения операций обновления распределенной базы данных (INSERT, UPDATE, DELETE), не разрушающее целостность и согласованность данных. Эта цель достигается применением двухфазового или двухфазного протокола фиксации транзакций (two-phase commit protocol), ставшего фактическим стандартом обработки распределенных транзакций. Его применение гарантирует согласованное изменение данных на нескольких узлах в рамках распределенной (или, как ее еще называют, глобальной) транзакции.

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

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 может осуществляться по сети. Спектр поддерживаемых конкретной СУБД сетевых протоколов не должен быть ограничением системы с распределенными базами данных. Данное качество формулируется максимально широко - в распределенной системе возможны любые сетевые протоколы.

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

Исходя из определения Дэйта, можно рассматривать DDB как слабосвязанную сетевую структуру, узлы которой представляют собой локальные базы данных. Локальные базы данных автономны, независимы и самоопределены; доступ к ним обеспечиваются СУБД, в общем случае от различных поставщиков. Связи между узлами - это потоки тиражируемых данных. Топология DDB варьируется в широком диапазоне - возможны варианты иерархии, структур типа "звезда" и т.д. В целом топология DDB определяется географией информационной системы и направленностью потоков тиражирования данных.

Становление систем управления базами данных совпало по времени со значительными успехами в развитии технологий распределенных вычислений и параллельной обработки. В результате возникли системы управления распределенными базами данных и системы управления параллельными базами данных. Именно эти системы становятся доминирующими инструментами для создания приложений интенсивной обработки данных.

5.1 Распределенные и параллельные базы данных

Благодаря интеграции рабочих станций в распределенную среду становится возможным более эффективное распределение функций в ней, когда прикладные программы выполняются на рабочих станциях, называемых серверами приложений, а базы данных обслуживаются выделенными компьютерами, называемыми серверами баз данных. Это послужило источником развития таких распределенных архитектур, где в роли узлов выступают не просто компьютеры общего назначения, а специализированные серверы.

Параллельный компьютер или мультипроцессор сам по себе является распределенной системой, составленной из узлов (процессоров, компонентов памяти), соединенных быстрой сетью внутри общего корпуса. Технологии распределенных баз данных могут быть естественным образом пересмотрены и распространены на системы параллельных баз данных, т. е. баз данных на параллельных компьютерах. Благодаря применяемому в системах этого типа параллелизму управления данными пользователи получают серверы баз данных высокой производительности и высокой доступности за существенно меньшую цену, чем эквивалентные системы на основе мэйнфреймов.

Таким образом, распределенная база данных (distributed database, DDB) – это совокупность логически взаимосвязанных баз данных, распределенных в компьютерной сети. Распределенная система управления базой данных определяется как программная система, которая позволяет управлять распределенной базой данных таким образом, чтобы ее распределенность была прозрачна для пользователей. В этом определении следует уточнить две отличительных архитектурных особенности. Первая из них заключается в том, что система состоит из (возможно, пустого) множества узлов приема запросов (query site) и непустого множества узлов данных (data site). Узлы данных обладают средствами для хранения данных, а узлы приема запросов – нет. В

узлах приема запросов лишь выполняются программы, реализующие пользовательский интерфейс для доступа к данным, хранящимся в узлах данных. Вторая особенность состоит в том, что узлы логически представляют собой независимые компьютеры. Следовательно, у такого узла имеется собственная основная и внешняя память, установлена собственная операционная система (может быть, одна и та же на всех узлах, а возможно, и нет) и имеется возможность выполнять приложения. Узлы связаны компьютерной сетью, а не входят в мультипроцессорную конфигурацию. Важно подчеркнуть слабую связанность процессоров, которые обладают собственными операционными системами и функционирует независимо.

База данных физически распределяется по узлам данных на основе фрагментации и репликации данных. При наличии схемы реляционной базы данных каждое отношение фрагментируется на горизонтальные или вертикальные разделы. Горизонтальная фрагментация реализуется при помощи операции селекции, которая направляет каждый кортеж отношения в один из разделов, руководствуясь предикатом фрагментации. Например, для отношения Employee возможна фрагментация в соответствии с местоположением рабочих мест служащих. При вертикальной фрагментации отношение делится на разделы при помощи операции проекции. Например, один раздел отношения Employee может содержать поля Emp_number, Emp_name и Address, а другой – поля Emp_number, Salary и Manager. За счет фрагментации данные приближаются к месту их наиболее интенсивного использования, что потенциально снижает затраты на пересылки; уменьшаются также размеры отношений, участвующих в пользовательских запросах.

Фрагменты данных могут также реплицироваться на основе характера доступа к ним. Это полезно, если доступ к одним и тем же данным производится из приложений, выполняющихся на разных узлах. В таком случае,

с точки зрения экономии затрат, более эффективно дублировать данные в ряде узлов, чем непрерывно пересылать данные между узлами.

При ослаблении отличительных особенностей распределенной СУБД получается параллельная система баз данных. Не существует четкого разграничения между параллельными и распределенными СУБД. В частности, архитектуры параллельных СУБД без совместно используемых ресурсов (sharing-nothing), которые обсуждаются ниже, схожи со слабо связанными распределенными системами. В параллельных СУБД используются новейшие многопроцессорные архитектуры, и на основе этого подхода создаются высокопроизводительные серверы баз данных высокой доступности, стоимость которых значительно ниже эквивалентных систем на мэйнфреймах.

Параллельную СУБД можно определить как СУБД, реализованную на мультипроцессорном компьютере. Такое определение подразумевает наличие множества альтернатив, спектр которых варьируется от непосредственного переноса существующих СУБД с переработкой лишь интерфейса к операционной системе до изощренных комбинаций алгоритмов параллельной обработки и функций баз данных, приводящих к новым аппаратно-программным архитектурам. Как и всегда, приходится выбирать между переносимостью (на несколько платформ) и эффективностью. Изощренные подходы направлены, главным образом, на более полное использование преимуществ конкретного мультипроцессора в ущерб переносимости.

Решение, тем самым, заключается в применении широкомасштабного параллелизма, чтобы усилить мощность отдельных компонентов путем их интеграции в целостную систему на основе соответствующего программного обеспечения параллельных баз данных. Важное значение имеет применение стандартных аппаратных компонентов, для того чтобы иметь возможность с минимальным отставанием использовать результаты постоянных технологических усовершенствований. В программном обеспечении базы

данных могут быть предусмотрены три вида параллелизма, присущие приложениям интенсивной обработки данных. Межапросный параллелизм предполагает одновременное выполнение множества запросов, относящихся к разным транзакциям. Под внутризапросным параллелизмом понимается одновременное выполнение сразу нескольких операций (например, операций выборки), относящихся к одному и тому же запросу. И внутризапросный, и межапросный параллелизм реализуется на основе разделения данных, аналогичного горизонтальному фрагментированию. Наконец, понятие внутриоперационного параллелизма означает параллельное выполнение одной операции в виде набора субопераций с применением, в дополнение к фрагментации данных, также и фрагментации функций. Языки баз данных, ориентированные на операции над множествами, обеспечивают много возможностей для использования внутриоперационного параллелизма.

Ниже перечислены характерные черты параллельных и распределенных СУБД:

1. Распределенная/параллельная база данных – это именно база данных, а не "коллекция" файлов, индивидуально хранимых на разных узлах сети. В этом заключается разница между DDB и распределенной файловой системой. Распределенные данные представляют собой DDB, только если они связаны в соответствии с некоторым структурным формализмом (таким как реляционная модель), а для доступа к ним имеется единый высокоуровневый интерфейс.
2. Система обладает полной функциональностью СУБД. Она не сводится по своим возможностям ни к распределенным файловым системам, ни к системам обработки транзакций. Обработка транзакций – только одна из функций, предоставляемых подобными системами. Наряду с этим они должны также обеспечивать функции запросов и структурной организации

данных, которые необязательно поддерживаются системами обработки транзакций.

3. Распределение (включая фрагментацию и репликацию) данных по множеству узлов невидимо для пользователей. Это свойство называется прозрачностью. Технология распределенных/параллельных баз данных распространяет основополагающую для управления базами данных концепцию независимости данных на среду, где данные распределены и реплицированы по множеству компьютеров, связанных сетью. Это обеспечивается за счет нескольких видов прозрачности: прозрачность сети (следовательно, прозрачность распределения), прозрачность репликации и прозрачность фрагментации. Прозрачность доступа означает, что пользователи имеют дело с единым логическим образом базы данных и осуществляют доступ к распределенным данным точно так же, как если бы они хранились централизованно. В идеале полная прозрачность подразумевает наличие языка запросов к распределенной СУБД, не отличающегося от языка для централизованной СУБД.

Вопросы прозрачности более критичны для распределенных, чем для параллельных СУБД. Для этого есть две причины. Во-первых, многопроцессорные системы, для которых реализуются параллельные СУБД, функционируют под управлением единой операционной системы. Операционная система может быть организована таким образом, чтобы брать на себя некоторые аспекты функциональности СУБД, предоставляя тем самым определенную степень прозрачности. Во-вторых, разработки программного обеспечения на параллельных системах поддерживаются языками параллельного программирования, также обеспечивающими некоторую степень прозрачности.

В распределенных СУБД данные и приложения, которые осуществляют доступ к ним, могут быть локализованы на одном и том же узле, благодаря чему

исключается (или сокращается) потребность в удаленном доступе к данным, характерная для систем телеобработки данных в режиме разделения времени. Далее, поскольку на каждом узле выполняется меньше приложений и хранится меньшая порция базы данных, можно сократить также конкуренцию при доступе к данным и ресурсам. Наконец, параллелизм, внутренне присущий распределенным системам, открывает возможности для реализации межзапросного и внутрizaпросного параллелизма.

Если доступ пользователей к базе данных заключается только в выполнении запросов (т. е. имеет место доступ только по чтению), то реализация межзапросного и внутрizaпросного параллелизма подразумевает реплицирование по возможности максимальной части базы данных. Но, поскольку на практике доступ к базе данных осуществляется не только по чтению, для реализации перемежающихся операций чтения и модификации данных необходима поддержка распределенных транзакций (обсуждаемых в одном из последующих разделов).

Высокая производительность – одна из важнейших целей, на достижение которой направлены технологии параллельных СУБД. Как правило, она обеспечивается за счет сочетания нескольких взаимно дополняющих решений, таких как применение операционных систем, ориентированных на поддержку баз данных, параллелизм, оптимизация, балансировка нагрузки. Наличие операционной системы, "осведомленной" о специфических потребностях баз данных (например, относительно управления буферами), упрощает реализацию функций баз данных нижнего уровня и способствует снижению их стоимости. Так, затраты на передачу сообщения могут быть значительно снижены (до нескольких сот инструкций) за счет применения специализированного коммуникационного протокола. Механизмы распараллеливания способствуют повышению общей пропускной способности системы (межзапросный

параллелизм), снижению времени отклика для отдельных транзакций (внутризапросный и внутриоперационный параллелизм).

Технологии распределенных и параллельных СУБД направлены также на повышение надежности, поскольку, благодаря репликации данных, исключаются одиночные точки отказа. Отказ одного узла или сбой на линии связи не приводит к выходу из строя всей системы. Даже если часть данных становится недоступной, при правильной организации системы пользователи могут иметь доступ к остальной части информации. Под "правильной организацией" понимается поддержка распределенных транзакций и протоколов обеспечения надежности (т. е. протоколов фиксации и восстановления).

В среде параллельных и распределенных СУБД упрощается решение вопросов, связанных с возрастанием объема баз данных или потребностей обработки. При этом редко возникает необходимость в серьезной перестройке системы; расширение возможностей обычно достигается за счет добавления процессорных мощностей или памяти.

В идеале параллельная (и, в меньшей степени, распределенная) СУБД обладает свойством линейной масштабируемости (linear scaleup) и линейного ускорения (linear speedup). Под линейной масштабируемостью понимается сохранение того же уровня производительности при увеличении размера базы данных и одновременном пропорциональном увеличении процессорной мощности и объема памяти. Линейное ускорение означает, что с наращиванием процессорной мощности и объема памяти при сохранении прежнего размера базы данных пропорционально возрастает производительность. Более того, при расширении системы должна потребоваться лишь минимальная реорганизация существующей базы данных.

С учетом соотношения цена/производительность для микропроцессоров и рабочих станций оказывается экономически выгоднее составить систему из

нескольких небольших компьютеров, чем реализовать ее на эквивалентной по мощности одной большой машине. Множество коммерческих распределенных СУБД функционирует на мини-компьютерах и рабочих станциях именно по причине более выгодного соотношения цена/производительность. Технологии, основанные на применении рабочих станций, получили столь широкое распространение благодаря тому, что большинство коммерческих СУБД способно работать в рамках локальных сетей, где в основном и используются рабочие станции. Развитие распределенных СУБД, предназначенных для глобальных сетей WAN, может привести к повышению роли мэйнфреймов. С другой стороны, распределенные СУБД будущих поколений, скорее всего, будут поддерживать иерархические сетевые структуры, узлы которых являются кластерами компьютеров, взаимодействующих в локальной сети, а сами кластеры соединяются посредством высокоскоростных магистралей.

При интеграции распределенных баз данных возникает вопрос однородности систем. Выделяют однородные и неоднородные распределенные базы данных. В неоднородных РБД используются различные СУБД. Основная проблема при этом заключается в сложности их интеграции. Остановимся на этом аспекте более подробно.

5.2 Однородные и неоднородные распределенные СУБД

Однородные распределенные системы баз данных относительно просты для понимания. Они имеют в своей основе один продукт СУБД, обычно с единственным языком баз данных (например, SQL с расширениями для управления распределенными данными). СУБД с поддержкой однородного распределения являются сильно связанными системами, их встроенные средства поиска данных и средства обработки запросов оптимизированы и настроены для достижения максимальной производительности и пропускной способности. На рисунке 5.1 изображена структура типичной однородной среды распределенной базы данных.

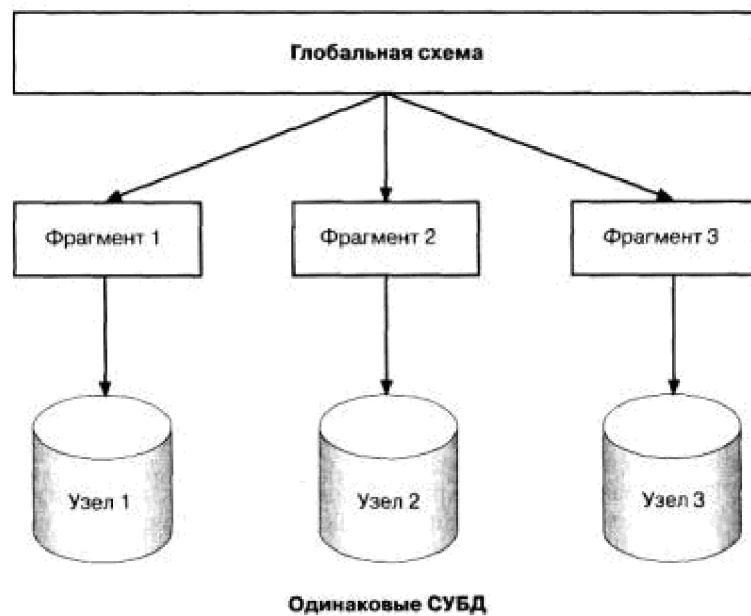


Рисунок 5.1 - Архитектура однородной распределенной базы данных

Существует множество вариантов однородной системы РБД. Так, на некотором узле может существовать одна глобально доступная "главная машина СУБД", с которой связаны компоненты для доступа к данным локальных баз данных, размещенные совместно с самими этими базами данных в пределах всей компании (или отдельного ее подразделения в зависимости от масштаба распределения). Более сложные модели могут допускать распределенность самой СУБД, когда каждый ее компонент на равных правах имеет доступ к данным любого другого узла. Однако относительно собственно управления данными мы имеем здесь идентичные модели хранения, структуры индексирования и форматы данных в рамках всей распределенной среды.

Противоположностью однородных систем РБД являются, конечно, неоднородные распределенные системы баз данных. Неоднородные системы включают два или более существенно различающихся продукта управления данными (например, реляционные СУБД от разных поставщиков, таких, как Oracle и Microsoft SQL Server, или СУБД одного поставщика, но

функционирующие на разных платформах и использующие различные структуры баз данных, такие, как DB2 и SQL/DS компании IBM). На рисунке 5.2 показана типичная конфигурация неоднородной распределенной базы данных.

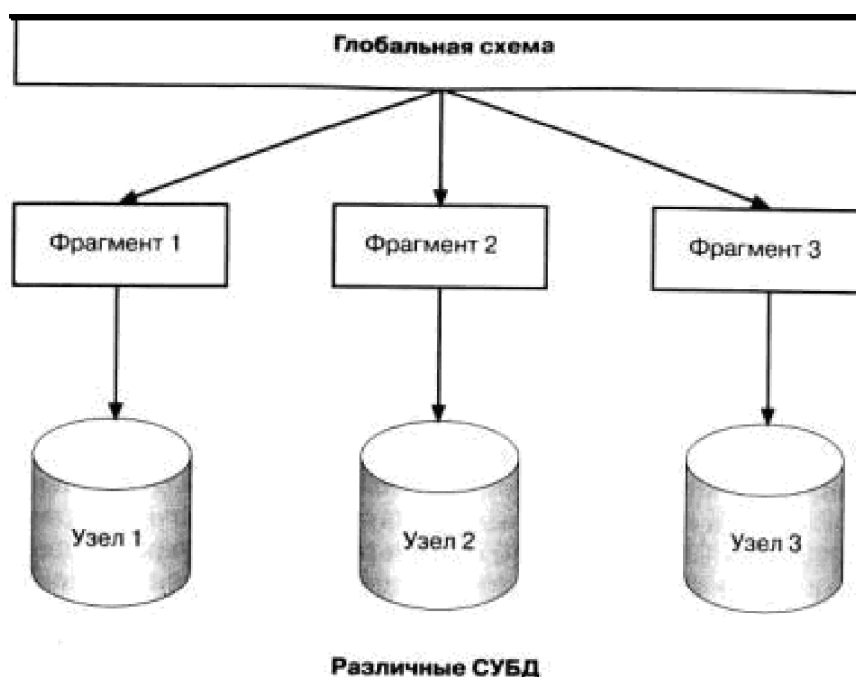


Рисунок 5.2 - Простая конфигурация неоднородной распределенной базы данных

Неоднородные системы баз данных можно, в свою очередь, также подразделить на классы в широком диапазоне - от федеративных систем до различных типов систем мультибаз данных; существует и формальная таксономия неоднородных моделей.

Однородные распределенные системы баз данных обычно проектируются методом "сверху вниз"; неоднородные же, напротив, чаще всего строятся "снизу вверх" с целью создать общую среду управления над существовавшими ранее разрозненными информационными ресурсами.

Рассмотрим сначала процесс построения распределенных баз данных методом "сверху вниз", поскольку он концептуально наиболее прост для понимания. Проектирование РБД "сверху вниз" осуществляется в целом аналогично проектированию централизованных баз данных. В идеале оно проводится с помощью одной из формальных методологий, которые включают создание концептуальной модели базы данных, отображение ее в логическую модель данных и, наконец, создание (и настройку) специфических для конкретной СУБД структур (например, таблиц базы данных СУБД Oracle).

Однако при проектировании РБД методом "сверху вниз" предполагается, что ее объекты не будут сосредоточены в одном месте, а распределяться по нескольким вычислительным системам (рисунок 5.3).

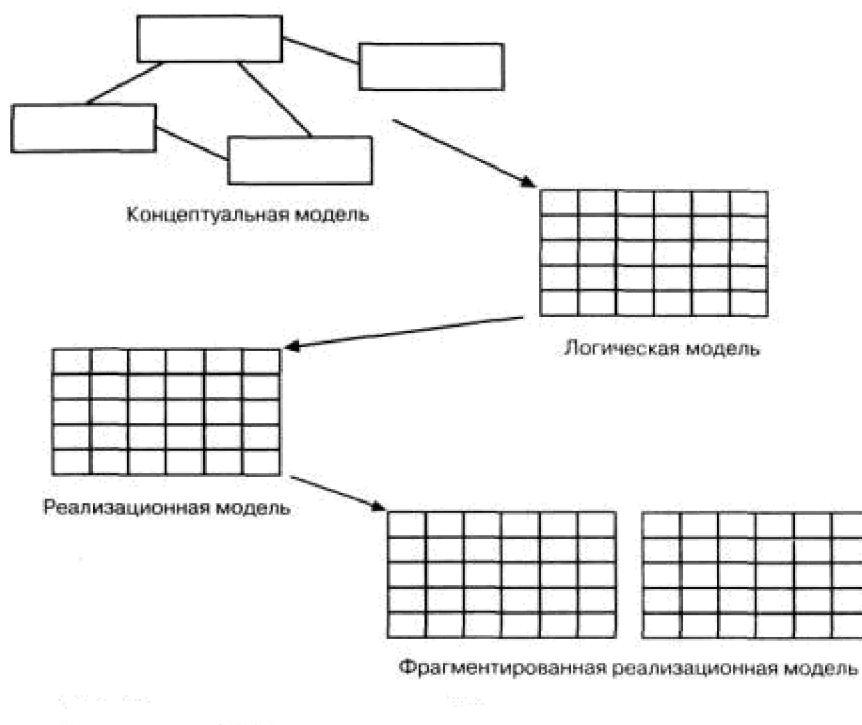


Рисунок 5.3 - Построение распределенной базы данных методом «сверху-вниз»

Распределение данных в РБД проводится путем фрагментации или тиражирования. Рассмотрим эти способы более подробно.

5.3 Вертикальная и горизонтальная фрагментация данных в РБД

Фрагментация означает декомпозицию объектов базы данных, таких, как реляционные таблицы, на две или более частей, которые размещаются на разных компьютерных системах. Классический пример, который обычно используют для иллюстрации этого понятия, - таблица с данными о сотрудниках или о заказах на продажу, разделенная на фрагменты по географическому или другому характеристическому признаку.

На рисунке 5.4 показана горизонтальная фрагментация, когда в таблице делаются горизонтальные "срезы" в соответствии со значением, скажем, какого-либо столбца таблицы. Строки данных о сотрудниках могут разбиваться на подмножества, соответствующие филиалам. Данные о продажах фрагментируются по магазинам, где эти продажи производились. Фрагментация может осуществляться также по принципу "карусели" (round-robin), а не на основе значений данных.

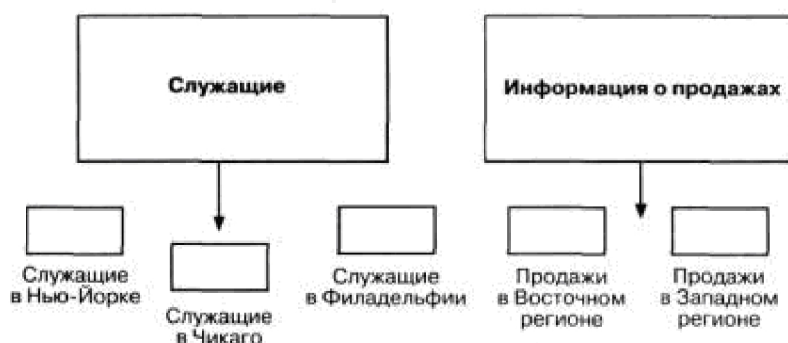


Рисунок 5.4 - Горизонтальная фрагментация

Альтернативная модель фрагментации - вертикальная - означает разбиение таблицы не по строкам, а по столбцам (рисунок 5.5). В этом случае некоторая часть информации о каждом сотруднике хранится в одном месте, а другая часть (относящаяся к той же таблице) - в другом.



Рисунок 5.5 - Вертикальная фрагментация

Независимо от того, применяется горизонтальная или вертикальная фрагментация, поддерживается глобальная схема, позволяющая воссоздать из имеющихся фрагментов логически централизованную таблицу или другую структуру базы данных.

Прежде чем перейти к обсуждению тиражирования, обратимся к практическим аспектам применения фрагментации в реальных базах данных.

На первый взгляд приведенные выше классические примеры интегрированных, сконструированных "сверху вниз" с последующей фрагментацией РБД могут показаться вполне осмысленными, но на самом деле они весьма далеки от жизни, по крайней мере в том виде, как они были представлены. Для среды приложения, структура которой показана на рисунке 5.4, разумными, по-видимому, были бы операции типа:

- обработка платежной ведомости для сотрудников отделения в Тусоне с помощью приложений, выполняемых там же;
- обработка платежной ведомости для сотрудников отделения в Нью-Йорке с помощью приложений, выполняемых в Нью-Йорке;
- запрос статистики о прокате и продажах видеопродукции по всем отделениям компании, выполняемый менеджерами отделения в Тусоне, обладающими необходимыми полномочиями.

Идея создания глобальной схемы, поддерживаемой над фрагментированными таблицами, кажется вполне разумной для реализации подобных задач. Однако нужно учесть следующее:

- При обработке платежной ведомости для сотрудников Тусонского отделения приложению, выполняющемуся в этом отделении, нет необходимости обращаться к какой-либо глобальной схеме. Такому приложению вовсе не нужно выполнять выборку вида:

```
SELECT *  
FROM ALL_EMPLOYEES_EVERYWHERE  
WHERE JOB_LOC = "Tucson"
```

где ALL_EMPLOYEES_EVERYWHERE - таблица с данными обо всех сотрудниках компании; JOB_LOC - столбец с названиями отделений. Для решения такой задачи достаточно автономной обработки локально хранимых данных.

- Менеджеру Тусонского отделения практически никогда не приходится обновлять данные о сотрудниках Нью-йоркского отделения (или других отделений), т. е. изменение персональной информации на удаленном узле при посредстве глобальной схемы - случай редкий и, вероятно, практически нереальный.
- Точно так же нью-йоркскому менеджеру нет нужды изменять данные о прокате или продажах видеопродукции в Тусоне. Разумеется, региональному менеджеру или представителю руководства компании, работающему в Нью-Йорке, могут понадобиться средства выполнения запросов только на чтение для выборки данных о продажах в Тусоне, чтобы затем соединить эти данные с информацией из Чикаго, Далласа и других отделений с целью проведения сравнительного анализа или получения сводного отчета. В то же время гранулярность данных о продажах, которые будут присланы такому

приложению (или другим подобным приложениям), будет значительно "тоньше", чем это реально необходимо.

Представителю центрального руководства вряд ли понадобятся полные данные о продажах или прокате каждой видеокассеты в каждом из магазинов. Его, скорее всего, будет интересовать некоторая обобщенная информация (например, объем продаж по городу за отчетный месяц в сравнении с предыдущим месяцем или с тем же месяцем предыдущего года и т.п.).

Очевидно, что в развитой коммерческой организации получение подобных отчетов наверняка будет предусмотрено. Тем не менее, можно не без оснований утверждать, что модель данных с фрагментацией и средствами воссоздания глобальных данных - не лучший способ достижения таких целей. В сфере управления информацией сейчас активно формируется школа, сторонники которой считают, что необходимо проводить различие между операционными базами данных и базами данных, предназначенными для поддержки принятия решений и других подобных целей. Одна из парадигм, относящихся к этому направлению - хранилища данных.

С учетом всего сказанного можно сделать вывод: идея глобальной схемы над распределенными фрагментированными данными в действительности не настолько привлекательна, как принято, было считать. Операции модификации данных на самом деле кластеризуются относительно локально управляемых данных. Динамические выборки в среде сильно распределенных данных могут поглотить львиную долю производительности операционной системы из-за необходимости многочисленных пересылок больших объемов данных с одного узла на другой (или другие) только для того, чтобы получить итоговую информацию. Хотя в однородных РБД, спроектированных "сверху вниз" с распределением данных путем фрагментации, возможна "интеллектуализация" формирования планов обработки запросов (предусматривающая, например, выполнение группирования на удаленном узле и пересылку только обобщенных

результатов), однако полезность этого подхода к задачам реального управления распределенной информацией остается под вопросом.

Означает ли это, что реализация однородных распределенных баз данных - идея совершенно нежизнеспособная? Отнюдь нет. Параллельные системы баз данных - это, возможно, одна из наиболее многообещающих моделей фрагментации в аспекте достижения высокой производительности и пропускной способности.

5.4 Репликация (тиражирование) данных в РБД

Обратимся теперь ко второму способу распределения данных, а именно к тиражированию. Тиражирование (или репликация) означает создание дубликатов данных.

Репликации - это множество различных физических копий некоторого объекта базы данных (обычно таблицы), для которых в соответствии с определенными в базе данных правилами, поддерживается синхронизация (идентичность) с некоторой "главной" копией. Теоретически значения всех данных в тиражированных объектах должны автоматически и незамедлительно синхронизироваться друг с другом. На практике это правило обычно несколько ослабляется. В некоторых системах копии используются исключительно в режиме чтения и обновляются в соответствии с заданным расписанием. В других средах допускается модификация отдельных значений в копиях, и эти изменения распространяются в соответствии с процедурами планирования и координации. На рисунке 5.6 показаны различные модели тиражирования.

До сих пор наше внимание в этом разделе было сосредоточено на подходе, предусматривающем построение распределенных баз данных "сверху вниз" с применением фрагментации или тиражирования. Повторим, что эта идеология применима только к однородным РБД, для которых вначале определяется глобальная схема, а затем производится распределение объектов базы данных.



Рисунок 5.6 - Модели тиражирования данных

Такой подход оправдан при создании новых приложений, но гораздо вероятнее, что вашей организации придется решать задачу создания интегрированной среды путем объединения существующих баз данных и соответствующих информационных менеджеров, возможно, в дополнение к некоторым вновь проектируемым компонентам баз данных.

В этом случае разработчики не могут позволить себе "роскошь" проектирования "сверху вниз". Здесь приходится прибегать к проектированию

"снизу вверх", где основной проблемой становится объединение схем уже существующих баз данных, чтобы предоставить как новым, так и прежним приложениям доступ и к новым, и к старым ресурсам данных (рисунок 5.7).

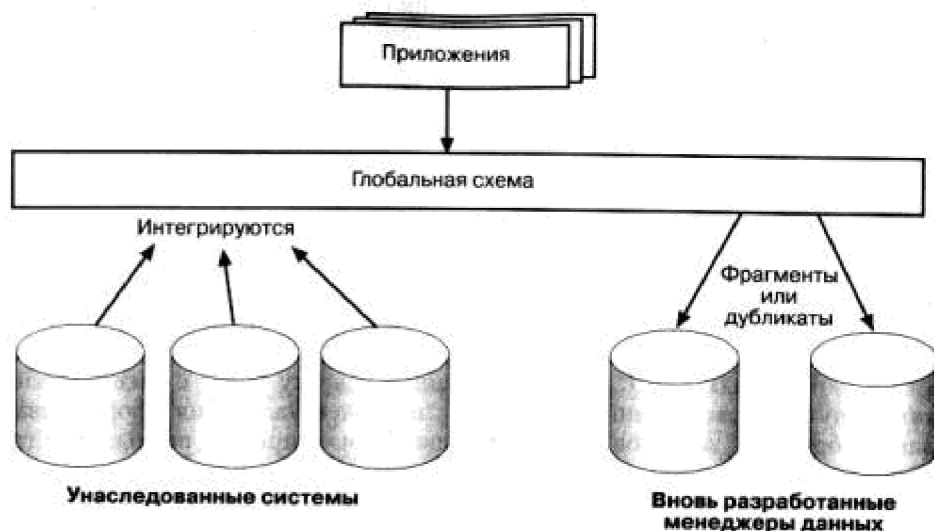


Рисунок 5.7 - Интеграция распределенных баз «снизу-вверх»

Среди многих сложных технических проблем, которые приходится при этом решать, отметим следующие:

- взаимное отображение различных моделей данных (т.е. наличие некоторого способа глобального доступа к множественным формам представления данных: к плоским файлам, к иерархическим, реляционным, объектно-ориентированным базам данных);
- управление метаданными;
- разрешение несоответствий импеданса (таких, как различные представления некоторого типа объекта данных в разных базах данных, когда, допустим, элемент MOVIE_TYPE - тип фильма - в одной базе данных имеет числовое представление, а в другой - символьное).

6 Перспективы распределенной обработки данных в современных СУБД

В последнее время возникло течение, авторитетные представители которого утверждают о необходимости радикальных перемен. Они утверждают, что «безразмерные» архитектуры универсальных систем управления данными, возраст которых измеряется десятками лет, не в состоянии удовлетворять потребности многих новых приложений. Они говорят, что наступает новая эпоха архитектурно упрощенных специализированных систем управления данными, предельно эффективно и экономично поддерживающих приложения соответствующих достаточно узких предметных областей.

Ряд экспериментальных и коммерческих разработок, которые подтверждают эти утверждения, говорят о наличии революционной ситуации в области управления данными и потребности в коренном отказе от традиционных архитектур СУБД.

Одним из основных событий 2008 г. явилась очередная встреча ведущих специалистов в области управления данными (из академических и коммерческих кругов), результатом которой явился «Клермонтский отчет» об исследованиях в области баз данных. В подобных отчетах анализируется текущее положение дел и предлагается программа исследований и разработок на ближайшие годы. В «Клермонтском отчете», также, отмечалась потребность в пересмотре архитектур систем управления данными, а также, в частности, указывалось на потребность исследований архитектур СУБД, предназначенных для поддержки «облачных» приложений.

В 2009 г. появилось несколько публикаций, свидетельствующих о ряде успешных разработок систем управления данными, которые основываются на новых архитектурах. В целом они демонстрируют некоторую общую тенденцию, которую стоит проанализировать специальным образом.

В 2006-2008 гг. разными авторами было опубликовано несколько статей, в которых с разной степенью убедительности утверждалось о необходимости радикальных перемен в области управления данными.

Основные положения указанных статей состоят в следующем:

- Архитектура современных SQL-ориентированных СУБД появилась более 30 лет тому назад, когда рынок систем управления данными был единым, не фрагментированным на специализированные секторы. В это время СУБД вынужденно делались «безразмерными», пригодными для использования в любой области приложений баз данных. Эта «безразмерность» присутствует сегодня в продуктах основных поставщиков. Плюсами основных SQL-ориентированных СУБД являются надежность и общая высокая производительность. Минусы – сложность, объемность и высокие накладные расходы, свойственные универсальности.
- За прошедшие 30 с лишним лет рынок систем управления данными сильно фрагментировался. Появились большие секторы рынка, для которых очень существенна высокая производительность приложений, которая не достигается или достигается с недопустимо большими затратами при использовании «безразмерных» СУБД. Экономически целесообразной стала разработка специализированных систем, которые ориентируются на эффективную поддержку заранее известных сценариев использования.
- За эти же тридцать лет в области управления данными была выполнена громадная исследовательская работа, результаты которой можно успешно применять для разработки специализированных систем.
- В связи с быстро меняющимися требованиями рынка успешными могут быть только такие новые продукты, которые можно вывести на рынок достаточно быстро – через год или два после начала разработки. Это еще один довод в пользу специализированных систем, нацеленных на решение узкого класса задач. В таких системах требуются более простые языковые средства,

упрощается оптимизация запросов и другие аспекты, являющиеся традиционным камнем преткновения «безразмерных» систем.

В одной из статей приводится краткая характеристика и показатели производительности экспериментальной системы ASAP, ориентированной на поддержку математических баз данных. Результаты тоже впечатляют, хотя опубликованных данных относительно общей организации и интерфейсов системы явно недостаточно, чтобы можно было хорошо понять принципы ее организации. Впрочем, это явно будет исправлено при разработке системы SciDB, которая основывается на ASAP.

Конечно, вызывает определенные сомнения как целесообразность и даже необходимость полномасштабного отказа от традиционных архитектур СУБД, так и некоторые технические аспекты конкретных систем и их характеристики, описываемые в упомянутых выше статьях. В частности, к настоящему времени накоплено так много технологических возможностей, аппаратных и программных, что большинство пользователей и разработчиков предпочитает потратить больше средств и/или пожертвовать некоторой долей производительности, чем перейти к использованию радикально других программных средств. Удачный опыт внедрения принципиально новых средств управления потоковыми данными не опровергает эти соображения. В этом случае речь идет об области приложений, которую принципиально не удовлетворяло существовавшее положение дел.

Таким образом, в качестве одной из наиболее актуальных тем исследований отмечается пересмотр архитектуры серверов баз данных. У современных развитых коммерческих систем реляционных баз данных имеются хорошо известные ограничения. Обеспечивая широкий набор возможностей, они показывают пиковую производительность только для очень ограниченного набора режимов. В последнее десятилетие появилось много популярных задач, связанных с обработкой больших объемов данных, для которых реляционные

СУБД обеспечивают плохое соотношение «цена/производительность», и при решении которых от использования РСУБД пришлось отказаться.

В настоящее время имеются два разных направления: расширение диапазона применимости универсальных систем баз данных и радикальное повышение производительности путем разработки специализированных систем баз данных для конкретных прикладных областей. У обоих направлений имеются свои достоинства, и очевидная общность конечных целей подсказывает, что работы в этих направлениях можно выполнять с взаимной пользой: специализированные методы можно повторно использовать в более универсальных системах, а использование архитектурных компонентов универсальных систем может позволить быстрее создавать прототипы новых специализированных систем.

На самом деле, так и происходит. Наиболее развитые традиционные СУБД непрерывно развиваются за счет внедрения технологий, разработанных для специализированных (часто экспериментальных) систем (заметим, что это отнюдь не снижает их уровень сложности), а при разработке специализированных систем с новой архитектурой грамотные разработчики стремятся к максимальному использованию проверенных методов.

К числу наиболее важных исследовательских тем в этой области относятся:

1. разработка систем для кластеров многоядерных процессоров, в которых имеется ограниченный и неоднородный доступ к памяти вне кристалла;
2. использование удаленной основной и флэш-памяти в качестве среды персистентного хранения данных в дополнение к памяти на магнитных дисках;
3. разработка унифицированного подхода к постоянно выполняемой адаптации и самонастройке оптимизации запросов и физических структур хранения данных;

4. сжатие и шифрование данных на уровне хранения, интегрированное со структурой хранения и оптимизацией запросов;
5. разработка систем, опирающихся на нереляционные модели данных, вместо того, чтобы с трудом адаптировать эти данные для хранения в таблицах;
6. нахождение компромиссов между согласованностью и доступностью для достижения лучшей производительности и масштабируемости до уровня тысяч машин;
7. разработка СУБД, учитывающих потребление энергии, которые ограничивают энергопотребление без ущерба для масштабируемости.

Совершенно очевидно, что бурное развитие технологии флэш-памяти и «твердотельных дисков», доступность таких носителей с емкостью в сотни гигабайт создает новые перспективы для организации систем управления данными.

Другими словами, имеются основания предполагать, что применение флэш-памяти может существенно изменить как архитектуру, так и номенклатуру систем управления данными.

Следующей важной темой современных исследований можно выделить декларативное программирование для новых платформ. Актуальность этой проблемы возрастает буквально экспоненциально, поскольку программисты нацеливаются на все более сложные среды, включая многоядерные микропроцессоры, распределенные службы и платформы «облачных вычислений» (cloud computing). Неквалифицированным программистам требуется возможность легко писать надежные программы, масштабируемые при возрастании числа процессоров, как в слабосвязанных, так и в сильносвязанных архитектурах.

Подходы, облегчающие и упрощающие программирование приложений, которые ориентированы на обработку данных, окажут серьезное влияние на программирование в целом.

В качестве первого примера подобного подхода можно рассмотреть MapReduce. В подходе Map-Reduce привлекает его простота; он основывается на языковых методах и методах распараллеливания по данным, известных в течение десятилетий. Для исследователей баз данных значимость подхода Map-Reduce состоит в том, что он демонстрирует преимущества программирования с распараллеливанием по данным новым классам разработчиков.

Этот подход можно рассматривать как первый шаг на пути упрощения параллельной обработки данных. Интересно также отметить лояльное отношение в MapReduce среди некоторых разработчиков новых систем баз данных.

Вторым примером является появление новых декларативных языков, основанных на Datalog. В ряде сценариев использование подобного декларативного языка позволяет на порядки величин сократить объем кода, одновременно обеспечивая его распределенное или параллельное исполнение. Как отмечается в отчете, группы, выполняющие соответствующие исследования, очень слабо скоординированы, и возрождение декларативных языков в новых контекстах пока происходит практически стихийным образом. Тем не менее, сам факт реинкарнации Datalog кажется крайне интересным и потенциально многообещающим.

Наконец, третий пример происходит из области программирования корпоративных приложений. Сближению сред программирования и управления данными, устранению проблемы потери соответствия (*impedance mismatch*) способствует возникновение новых языковых расширений, таких как LINQ и Ruby on Rails. Однако в этих пакетах пока серьезно не решается проблема программирования с использованием нескольких машин. Для корпоративных приложений ключевым решением распределенной разработки является разделение логики приложения и данных между несколькими «звеньями» («*tier*»): Web-клиентами, Web-серверами, серверами приложений и серверной

СУБД. Особо ценной здесь является независимость данных, позволяющая создавать программы без предварительного принятия долговременных решений о физическом размещении программ и данных в разных звеньях.

Как полагают авторы отчета, одним из существующих языков, который мог бы способствовать подобному декларативному программированию, является XQuery, поскольку формат XML часто используется на разных уровнях протоколов.

Облачные сервисы могут повысить эффективность поставщиков приложений за счет сокращения объема требуемых начальных вложений и сокращения стоимости владения. Доступен ряд разнообразных облачных сервисов, включая прикладные сервисы (salesforce.com), сервисы хранения (Amazon S3), вычислительные сервисы (Google App Engine, Amazon EC2), и сервисы данных (Amazon SimpleDB, Microsoft SQL Server Data Services, Google's Datastore).

Общей проблемой поставщиков облачных сервисов является потребность в компромиссе между функциональными возможностями и эксплуатационными расходами. Интерфейсы существующих облачных служб намного более ограничены, чем в традиционных системах баз данных. В них поддерживаются минималистские языки запросов и ограниченные гарантии согласованности данных. Это затрудняет программирование приложений, но позволяет создавать более предсказуемые службы, в которых обеспечиваются соглашения об уровне обслуживания, трудно достижимые для полнофункциональных служб данных на основе языка SQL.

Следующая проблема – достижение высокого уровня управляемости. Решение этой проблемы для систем в облачной среде осложняется ограниченным человеческим вмешательством, сильно изменчивыми рабочими нагрузками и разнообразием совместно используемых инфраструктур. При решении проблемы требуется учитывать наличие различных подходов к

виртуализации ресурсов – от виртуализации аппаратных средств до использования в службе данных одной СУБД для управления несколькими базами данных с разными схемами. Для достижения управляемости требуется развивать и совершенствовать методы самоуправления системами.

Серьезную проблему представляет масштабируемость будущих облачных систем управления данными. Технология, применяемая при разработке современных SQL-ориентированных СУБД, просто не позволяет масштабировать их до 1000 узлов. По всей видимости, в будущих системах нужно ограничивать требования к их транзакционности или учитывать семантику данных. Дополнительные исследования нужны для понимания реальной масштабируемости облачной инфраструктуры.

Наконец, при совместном использовании физических ресурсов в облачной инфраструктуре требуется обеспечение безопасности и конфиденциальности данных, которые не могут гарантироваться за счет наличия физического разграничения машин или сетей.

Как показывает последнее время, хотим мы этого или не хотим, в мире баз данных происходят серьезные изменения. Специализация систем управления данными, поиск их новых архитектур, более приспособленных к текущим реалиям, становится относительной нормой.

В любом случае, мы живем в очень интересное время, время перемен в области управления данными. Займет ли новый мир место старого, привычного и для многих вполне удобного мира, или же два мира будут мирно существовать, покажет наступающее новое десятилетие.

Список используемых источников

1. Дейт К.. Введение в системы баз данных. 8-е изд. – М.; СПб.: Вильямс, 2005.
2. Кузнецов С.Д. Основы баз данных. 2-е изд. Москва, Бином, 2007.
3. Кузнецов С.Д. Базы данных: языки и модели. Москва, Бином, 2008.
4. Таненбаум Э., Стеен М. Распределенные системы. Принципы и парадигмы. -СПб: Питер, 2003.
5. Кузнецов С.Д. Базы данных. Вводный курс (http://citforum.ru/database/advanced_intro/)
6. Кузнецов С.Д. Основы современных баз данных /Информационно-аналитические материалы Центра Информационных Технологий (<http://citforum.ru/database/osbd/contents.shtml>)
7. Кузнецов С.Д. Год эпохи перемен в технологии баз данных (<http://citforum.ru/database/articles/epoch/>)
8. Ривкин М. Тенденции развития универсальных коммерческих СУБД (<http://citforum.ru/database/articles/trends/>)
9. Карпова Т.С. Базы данных: модели, разработка, реализация (<http://www.intuit.ru/department/database/dbmdi/>)