1. Распределенные базы данных

1.1. Введение

Технология распределенных баз данных, получившая в настоящее время широкое распространение, способствует обратному переходу от централизованной обработки данных к децентрализованной. Создание технологии систем управления распределенными базами данных является одним из самых больших достижений в области баз данных.

Основной причиной разработки информационных систем, использующих базы данных, является стремление интегрировать все обрабатываемые в организации данные в единое целое и обеспечить к ним контролируемый доступ. Хотя такая интеграция способствует централизации, последняя не является самоцелью. Создание компьютерных сетей приводит к децентрализации обработки данных. Децентрализованный подход, по сути, отражает организационную структуру предприятия, логически состоящего из отдельных подразделений, отделов, групп и тому подобного, которые физически распределены по разным офисам, отделениям или филиалам, причем каждая отдельная единица имеет дело с собственным набором обрабатываемых данных [1, 2]. Разработка распределенных баз данных, отражающих организационные структуры предприятий, позволяет сделать данные, поддерживаемые каждым из существующих подразделений, общедоступными, обеспечив при этом их сохранение именно в тех местах, где они чаще всего используются. Подобный подход расширяет возможности совместного использования информации, одновременно повышая эффективность доступа к ней.

Распределенные системы решают проблему *островов информации*. Базы данных можно представить как некие электронные острова, представляющие собой отдельные, и в общем случае, труднодоступные места, подобные удаленным друг от друга островам. Такое положение может являться следствием географической разобщенности, несовместимости используемой архитектуры компьютеров, несовместимости используемых коммутационных протоколов и т.д. Интеграция отдельных баз данных в одно логическое целое способна изменить подобное положение дел.

1.2. Основные понятия и определения

Распределенная база данных – это набор логически связанных между собой разделяемых данных и их описаний, которые физически распределены в некоторой компьютерной сети.

Распределенная система управления базой данных (РСУБД) – это программная система, предназначенная для управления распределенными базами данных и позволяющая сделать распределенность информации прозрачной для конечного пользователя.

Распределенная система управления базами данных состоит из единой логической базы данных, разделенной на некоторое количество **фрагментов**. Каждый фрагмент базы данных хранится на одном или нескольких компьютерах (узлах, sites), которые соединены между собой коммуникационной сетью и каждый из которых работает под управлением отдельной СУБД. Любой пользователь может выполнить операции над данными на своем локальном узле точно так же, как если бы этот узел вовсе не входил в распределенную систему (что создает определенную степень локальной автономии). С другой стороны, любой узел способен обрабатывать данные, сохраняемые на других компьютерах сети.

Пользователи взаимодействуют с распределенной базой данных через приложения. **Локальные приложения** не требуют доступа к данным на других узлах, **глобальные приложения** требуют подобного доступа. В распределенной СУБД должно существовать хотя бы одно глобальное приложение, поэтому любая РСУБД должна иметь следующие особенности [2].

- Набор логически связанных разделяемых данных.
- Сохраняемые данные разбиты на некоторое количество фрагментов.
- Между фрагментами может быть организована репликация данных.
- Фрагменты и их реплики распределены по различным узлам.
- Узлы связаны между собой сетевыми соединениями.
- Работа с данными на каждом узле управляется СУБД.
- СУБД на каждом узле способны поддерживать автономную работу локальных приложений.

Поясним, что репликация заключается в поддержке актуальной копии (реплики) некоторого фрагмента базы данных на нескольких различных узлах.

Нет необходимости в том, чтобы на каждом из узлов системы существовала своя собственная база данных, что и показано на примере топологии РСУБД, представленной на рис. 1.1.

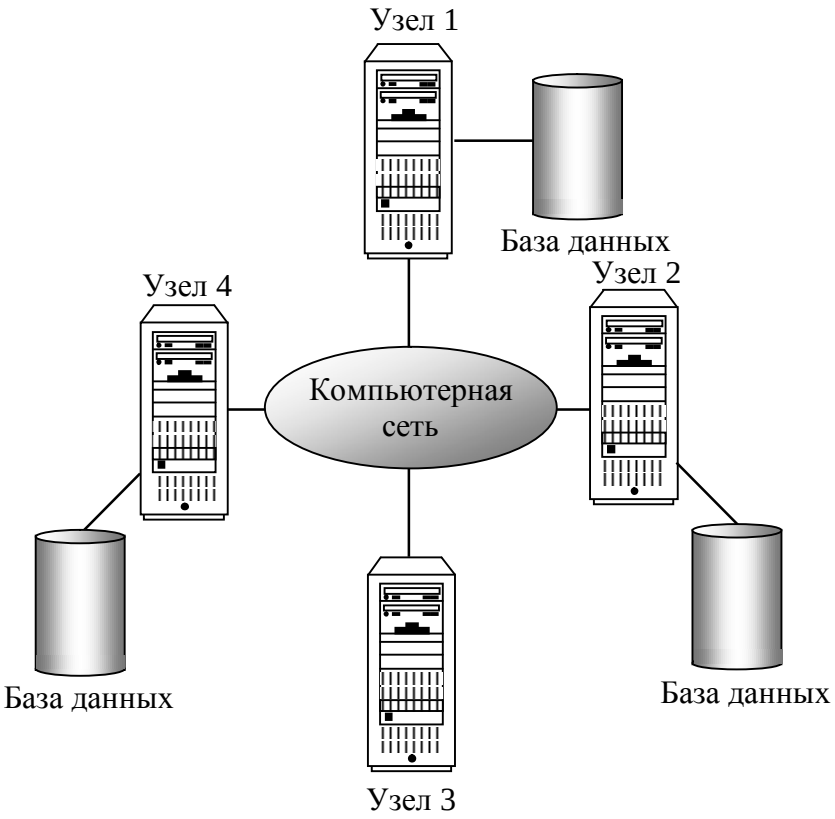


Рис. 1.1. Топология распределенной системы управления базой данных

Из определения РСУБД следует, что для конечного пользователя распределенность системы должна быть совершенно **прозрачна** (невидима). Другими словами, для пользователя распределенная система должна выглядеть так же, как нераспределенная система. В некоторых случаях это требование называют **фундаментальным принципом** построения распределенных СУБД [1].

1.2.1. Распределенная обработка

Рассмотрим различия, которые существуют между распределенными СУБД и распределенной обработкой данных.

Распределенная обработка – это обработка с использованием централизованной базы данных, доступ к которой может осуществляться с различных компьютеров сети.

Основополагающим моментом в определении распределенной базы данных является утверждение, что система работает с данными, *физически распределенными в сети*. Если данные хранятся централизованно, то даже в том случае, когда доступ к ним обеспечивается для любого пользователя в сети, данная система просто поддерживает распределенную обработку, но не может рассматриваться как распределенная СУБД. Схематически подобная топология представлена на рис. 1.2. Эта топология часто называется системой "клиент/сервер".

Рис. 1.2. Топология системы с распределенной обработкой

Точнее, система "клиент/сервер" – это система, в которой одни узлы – *клиенты*, а другие – *серверы*; все данные размещены на узлах, которые являются серверами; все приложения выполняются на узлах-клиентах и полная локальная независимость не предоставляется. Клиент – это процесс, посылающий запрос на обслуживание. Архитектура клиент-сервер обеспечивает прикладным программам клиента доступ к данным, которыми управляет сервер. Это основное назначение этой архитектуры, то есть несколько клиентов эффективно используют один сервер [3].

Возможны несколько вариантов основной схемы.

- Несколько клиентов могут совместно использовать один сервер.
- Отдельный клиент может иметь доступ к нескольким серверам. Такая возможность, в свою очередь, делится на два случая.
  a) Клиент ограничен доступом лишь к одному серверу за один раз, т.е. каждый отдельный запрос к базе данных должен быть ориентированным на один сервер. Невозможно в пределах одного запроса получить данные с двух или более различных серверов. Более того, пользователь должен знать, на каком именно сервере хранятся те или иные части данных.
  b) Клиент может иметь одновременный доступ к нескольким серверам, т.е. отдельный запрос может сочетать данные с нескольких серверов. А это означает, что несколько серверов предоставляются клиенту так, как будто это на самом деле один сервер. Пользователь не должен знать, какие части данных хранятся на каждом сервере.

Но в случае *b* фактически описан принцип системы распределенной базы данных. Это не совсем то, что подразумевают под термином "клиент/сервер" [1].

1.2.2. Параллельные СУБД

Остановимся на различиях, которые существуют между распределенными и параллельными СУБД.

Параллельная СУБД – это система управления базой данных, функционирующая с использованием нескольких процессоров и устройств жестких дисков, что позволяет ей распараллеливать выполнение

некоторых операций с целью повышения общей производительности обработки.

Применение параллельных СУБД позволяет объединить несколько маломощных машин для получения того же самого уровня производительности, что и в случае одной, но более мощной машины.

Для предоставления нескольким процессорам совместного доступа к одной и той же базе данных параллельная СУБД должна обеспечивать управление совместным доступом к ресурсам. То, какие именно ресурсы разделяются и как это разделение реализовано на практике, непосредственно влияет на показатели производительности создаваемой системы. К основным типам архитектуры параллельных СУБД относятся:

- системы с разделением памяти;
- системы с разделением дисков;
- системы без разделения.

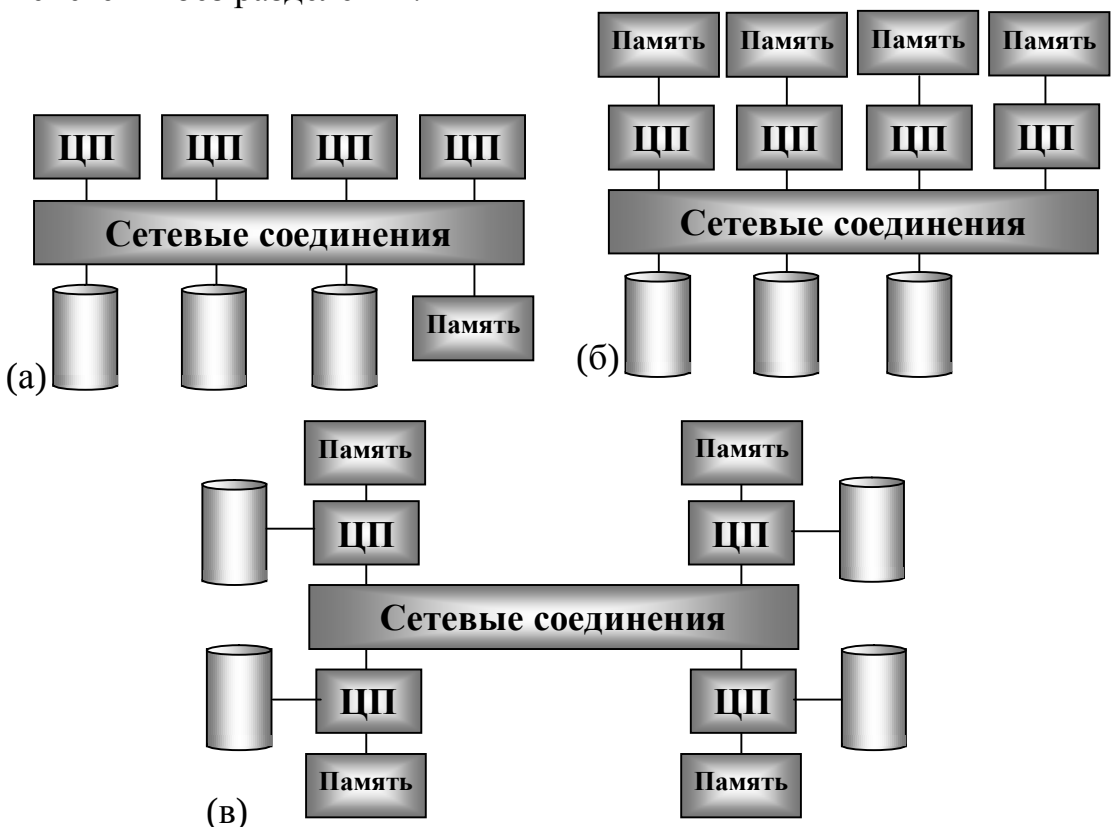


Рис. 1.3. Архитектура систем с параллельной обработкой:
а) с разделением памяти;
б) с разделением дисков;
в) без разделения

Хотя схему без разделения в некоторых случаях относят к распределенным СУБД, в параллельных системах размещение данных диктуется исключительно соображениями производительности. Более того, узлы (сайты) распределенной СУБД обычно разделены географически, независимо администрируются и соединены между собой относительно медленными сетевыми соединениями, тогда как узлы параллельной СУБД чаще всего располагаются на одном и том же компьютере или в пределах одного и того же сайта [2].

В состав **систем с разделением памяти** входит несколько процессоров, разделяющих общую системную память. Эту архитектуру называют также симметричной многопроцессорной обработкой. Она применяется для самых разных вычислительных платформ, начиная от персональных ЭВМ, содержащих несколько параллельно работающих микропроцессоров, больших RISC-систем и вплоть до крупнейших мейнфреймов. Эта архитектура обеспечивает быстрый доступ к данным для ограниченного числа процессоров (обычно не больше 64). При увеличении числа процессоров сетевые взаимодействия начинают ограничивать производительность всей системы.

Системы без разделения иначе называют системами массовой параллельной обработки. В таких системах каждый процессор имеет свою собственную оперативную и дисковую память. База данных распределена между всеми дисковыми устройствами вычислительных подсистем, связанных с этой базой данных. В результате все данные прозрачно доступны пользователям каждой из вычислительных подсистем. Такая архитектура обеспечивает более высокую масштабируемость, чем системы с разделением памяти, и позволяет легко организовать поддержку работы большого количества процессоров. Однако только в случае, когда требуемые данные хранятся локально, удается достичь максимальной производительности.

В **системах с разделением дисков** каждый из процессоров имеет непосредственный доступ ко всем совместно используемым дисковым устройствам, но обладает собственной оперативной памятью. Такие системы оптимальны для приложений с высокой централизованной обработкой и обеспечивают самые высокие показатели доступности и производительности. В системах с такой архитектурой, как и в случае архитек-

туры без разделения, исключаются узкие места, связанные с разделением памяти. Но в данном случае исключение таких узких мест происходит без дополнительной нагрузки по физическому распределению данных на разных устройствах. Разделяемые дисковые системы иногда называют *кластерами*.

Параллельные технологии обычно используются в системах, которые должны поддерживать выполнение тысяч транзакций в секунду, или в случае исключительно больших баз данных (несколько терабайт). Такие системы должны обеспечивать приемлемое время реакции на запрос и нуждаются в доступе к большому объему данных.

Вопросы

1.1. Поясните значение терминов «распределенная база данных» и «распределенная СУБД. Назовите причины создания подобных систем.

1.2. Перечислите особенности, которые должна иметь любая РСУБД.

1.3. Сравните и укажите отличия между РСУБД и системами с распределенной обработкой. При каких обстоятельствах выбор РСУБД оказывается предпочтительней организации распределенной обработки?

1.4. Сравните и укажите отличия между РСУБД и системами с параллельной обработкой. При каких обстоятельствах РСУБД оказывается предпочтительнее параллельной СУБД?

1.3. Гомогенные и гетерогенные распределенные СУБД

Распределенные СУБД классифицируются как гомогенные и гетерогенные [2]. Если все узлы распределенной системы используют один и тот же тип СУБД, то такая система называется **гомогенной**. В **гетерогенных** системах на сайтах могут функционировать различные типы СУБД, использующие разные модели данных (реляционные, сетевые, иерархические и др.).

Гомогенные системы проще проектировать и сопровождать. Такие РСУБД позволяют поэтапно наращивать размеры системы, добавляя но-

вые узлы к уже существующей распределенной системе. Более того, организуя на различных узлах параллельную обработку информации, можно повысить производительность всей системы.

Гетерогенные системы возникают в случаях интеграции во вновь создаваемую распределенную систему независимых узлов со своими собственными системами баз данных. В гетерогенных системах для организации взаимодействия между различными типами СУБД потребуется организовать трансляцию передаваемых сообщений. Пользователи каждого из узлов должны иметь возможность вводить свои запросы на языке той СУБД, которая используется на этом узле (прозрачность в отношении используемой СУБД). Система должна обеспечить локализацию требуемых данных и выполнение трансляции передаваемых сообщений. В общем случае данные могут быть запрошены с другого узла, который имеет:

- иной тип используемого оборудования;
- иной тип используемой СУБД;
- иной тип применяемых оборудования и СУБД.

Если используется другой тип оборудования, но та же СУБД, методы выполнения трансляции заключаются в замене кодов и изменении длины слова. Если типы используемых на узлах СУБД различны, трансляция усложняется необходимостью отображения структуры данных одной модели в соответствующие структуры данных другой модели. Например, отношения в реляционной модели данных должны быть преобразованы в записи и наборы, типичные для сетевой модели данных. Кроме того, необходимо транслировать текст запросов с одного используемого языка манипулирования данными на другой. Если отличаются и тип оборудования, и тип используемой СУБД, потребуется выполнять оба вида трансляции.

Дополнительные сложности возникают при попытке выработки единой концептуальной схемы, создаваемой путем интеграции независимых локальных концептуальных схем.

Типичное решение, применяемое в некоторых реляционных системах для обеспечения прозрачности в отношении используемой СУБД, состоит в использовании **шлюзов**. В состав отдельных частей гетерогенных распределенных систем должны входить шлюзы, предназначенные

для преобразования языка и модели данных каждого из используемых типов СУБД в язык и модель данных реляционной системы. Однако такой подход не свободен от некоторых серьезных недостатков. Например, шлюзы не позволяют организовать систему управления транзакциями даже для отдельных пар систем. То есть шлюз между двумя системами представляет собой не более чем транслятор запросов. Поэтому шлюзы не позволяют справиться с проблемами, вызванными неоднородностью структур и представлением данных в различных схемах.

Созданная рабочая группа (Specification Working Group – SWG) должна подготовить спецификации, регламентирующие инфраструктуру среды базы данных [2]:

- унифицированный и мощный интерфейс языка SQL (SQL API), позволяющий создавать клиентские приложения так, чтобы они не были привязаны к конкретному типу используемой СУБД;
- унифицированный протокол доступа к базе данных, позволяющий СУБД одного типа непосредственно взаимодействовать с СУБД другого типа, без необходимости использования какого-либо шлюза;
- унифицированный сетевой протокол, позволяющий осуществлять взаимодействие СУБД различного типа.

Наиболее важной задачей этой группы является поиск способа, позволяющего в одной транзакции выполнять обработку данных, содержащихся в нескольких базах, управляемых СУБД различных типов, причем без необходимости использования каких-либо шлюзов.

1.4. Мультибазовые системы

Одной из разновидностей распределенных СУБД являются мультибазовые системы.

Мультибазовая система – распределенная система управления базами данных, в которой управление каждым из узлов осуществляется совершенно автономно [2].

В мультибазовых системах предпринимается попытка интеграции таких распределенных систем баз данных, в которых весь контроль над отдельными локальными системами целиком и полностью осуществляет-

ся их операторами. Полная автономия узлов позволяет не вносить какие-либо изменения в локальные СУБД. Следовательно, мультибазовые СУБД требуют создания поверх существующих локальных систем дополнительного уровня программного обеспечения, предназначенного для предоставления необходимой функциональности.

Мультибазовые системы позволяют конечным пользователям разных узлов получать доступ и совместно использовать данные без необходимости физической интеграции существующих баз данных. Они обеспечивают пользователям возможность управлять базами данных их собственных узлов без какого-либо централизованного контроля, который обязательно присутствует в обычных типах РСУБД. Администратор локальной базы данных может разрешить доступ к определенной части своей базы данных посредством создания *схемы экспорта*, определяющей, к каким элементам локальной базы данных смогут получать доступ внешние пользователи.

Говоря простыми словами, мультибазовая СУБД является такой СУБД, которая прозрачным образом располагается поверх существующих баз данных и файловых систем, предоставляя их своим пользователям как некоторую единую базу данных. Такая поддержка глобальной схемы позволяет пользователям на основании этой схемы строить запросы и модифицировать данные. Мультибазовая СУБД работает только с глобальной схемой, тогда как локальные СУБД собственными силами обеспечивают поддержку данных всех их пользователей. Глобальная схема создается посредством интеграции схем локальных баз данных. Программное обеспечение мультибазовой СУБД предварительно транслирует глобальные запросы и превращает их в запросы и операторы модификации данных соответствующих локальных СУБД. Полученные после выполнения локальных запросов результаты сливаются в единый глобальный результат, предоставляемый пользователю. Кроме того, мультибазовая СУБД осуществляет контроль за выполнением фиксации или отката отдельных операций глобальных транзакций локальных СУБД, а также обеспечивает сохранение целостности данных в каждой из локальных баз данных. Программы мультибазовой СУБД управляют различными шлюзами, с помощью которых контролируют работу локальных СУБД.

1.5. Преимущества и недостатки распределенных СУБД

Основной причиной использования распределенных баз данных является то, что обычно предприятия уже распределены, по крайней мере, логически, т.е. на подразделения, отделы, рабочие группы и т.д. Крупные организации могут быть распределены и физически на отделения, заводы, лаборатории, которые могут находиться в разных концах страны и даже за ее пределами. Вполне логично будет предположить, что данные также распределены, поскольку каждая организационная единица создает и обрабатывает собственные данные, относящиеся к деятельности этой единицы. Таким образом, информация предприятия разбивается на части, которые можно назвать *островами информации* [1]. Распределенная база данных обеспечивает *мосты* для их соединения в целое. В подобной базе данных персонал отделения компании сможет выполнять необходимые ему локальные запросы. Руководству компании может потребоваться выполнять глобальные запросы, предусматривающие получение доступа к данным, хранящимся во всех отделениях компании. Иначе говоря, распределенная система позволяет структуре базы данных **отражать структуру организации**. Это является наиболее важным преимуществом распределенных СУБД.

В распределенных системах данные размещаются на том сайте, на котором зарегистрированы пользователи, которые их чаще всего используют. В результате пользователи этого узла получают локальный контроль над требуемыми им данными и могут регулировать локальные ограничения на их использование. В этом заключается **разделяемость и локальная автономность** распределенных СУБД.

В централизованных СУБД отказ центрального компьютера вызывает прекращение функционирования всей СУБД. Распределенные СУБД проектируются так, чтобы обеспечить работоспособность системы, несмотря на отказ одного из узлов РСУБД или линии связи между узлами. Это достигается организацией репликации данных, так что данные и их копии будут размещены на более чем одном сайте. Система будет перенаправлять запросы к отказавшему узлу в адрес другого сайта. Это приводит к **повышению надежности системы и доступности данных**.

В настоящее время считается, что намного дешевле собрать из небольших компьютеров систему, мощность которой будет эквивалентна мощности одного большого компьютера. Оказывается, что намного выгоднее устанавливать в подразделениях организации собственные маломощные компьютеры, кроме того, гораздо дешевле добавить в сеть новые рабочие станции, чем модернизировать систему с мейнфреймом. Из этого следуют **экономические преимущества** использования РСУБД.

Благодаря **модульности** распределенной среды расширение существующей системы осуществляется намного проще. Добавление в сеть нового узла не оказывает влияния на функционирование уже существующих. Подобная гибкость позволяет организации легко расширяться. В централизованных СУБД рост размера базы данных может потребовать замены и вычислительной системы на более мощную, и используемого программного обеспечения на более мощную и гибкую СУБД.

Распределенным системам свойственны и некоторые *недостатки*, наиболее существенным из которых является **повышение сложности**, по крайней мере, с технической точки зрения. Распределенные СУБД являются более сложными программными комплексами, чем централизованные СУБД. Достаточно указать на тот факт, что данные могут подвергаться репликации. Если репликация данных не будет поддерживаться на требуемом уровне, система будет иметь более низкий уровень доступности данных, надежности и производительности, чем централизованные системы.

В распределенных системах могут возникнуть **проблемы защиты** не только данных, реплицируемых на несколько различных сайтов, но и защиты сетевых соединений самих по себе. В централизованных системах доступ к данным легко контролируется.

Усложнение контроля за целостностью данных – еще один из недостатков распределенных СУБД. Требования обеспечения целостности (корректности и согласованности данных) формулируются в виде ограничений. Выполнение ограничений гарантирует защиту информации в базе данных от разрушения. Реализация таких ограничений целостности требует доступа к большому количеству данных, используемых во время проверок. В распределенных СУБД повышенная стоимость передачи и

обработки данных может препятствовать организации эффективной защиты от нарушений целостности данных.

Наконец, нельзя не сказать об **усложнении процедуры проектирования базы данных**. Помимо обычных проблем, связанных с проектированием централизованных баз данных, разработка распределенных СУБД требует принятия решения о фрагментации данных, распределения фрагментов по отдельным сайтам и организации процедур репликации данных.

Вопросы

1.5. В чем заключаются методы выполнения трансляции в гетерогенных РСУБД?

1.6. Каковы недостатки использования шлюзов для обеспечения прозрачности в отношении используемой СУБД?

1.7. Какие спецификации должна подготовить Specification Working Group?

1.8. Сравните и укажите отличия между обычными типами РСУБД и мультибазовыми системами. При каких обстоятельствах выбор мультибазовой системы оказывается предпочтительней организации обычной РСУБД?

1.9. Назовите преимущества и недостатки, свойственные распределенным системам.

1.6. Архитектура распределенных СУБД

Трехуровневая архитектура ANSI-SPARC для СУБД, обсуждавшаяся, например, в [3], представляет собой типовое решение для централизованных СУБД. Однако распределенные СУБД имеют множество отличий, которые достаточно сложно отобразить в некотором эквивалентном архитектурном решении, приемлемом для большинства случаев. Можно найти некоторое рекомендуемое решение, учитывающее особенности работы с распределенными данными. Один из примеров рекомендуемой архитектуры РСУБД приведен в [2] и представлен на рис. 1.4. Он включает следующие элементы:

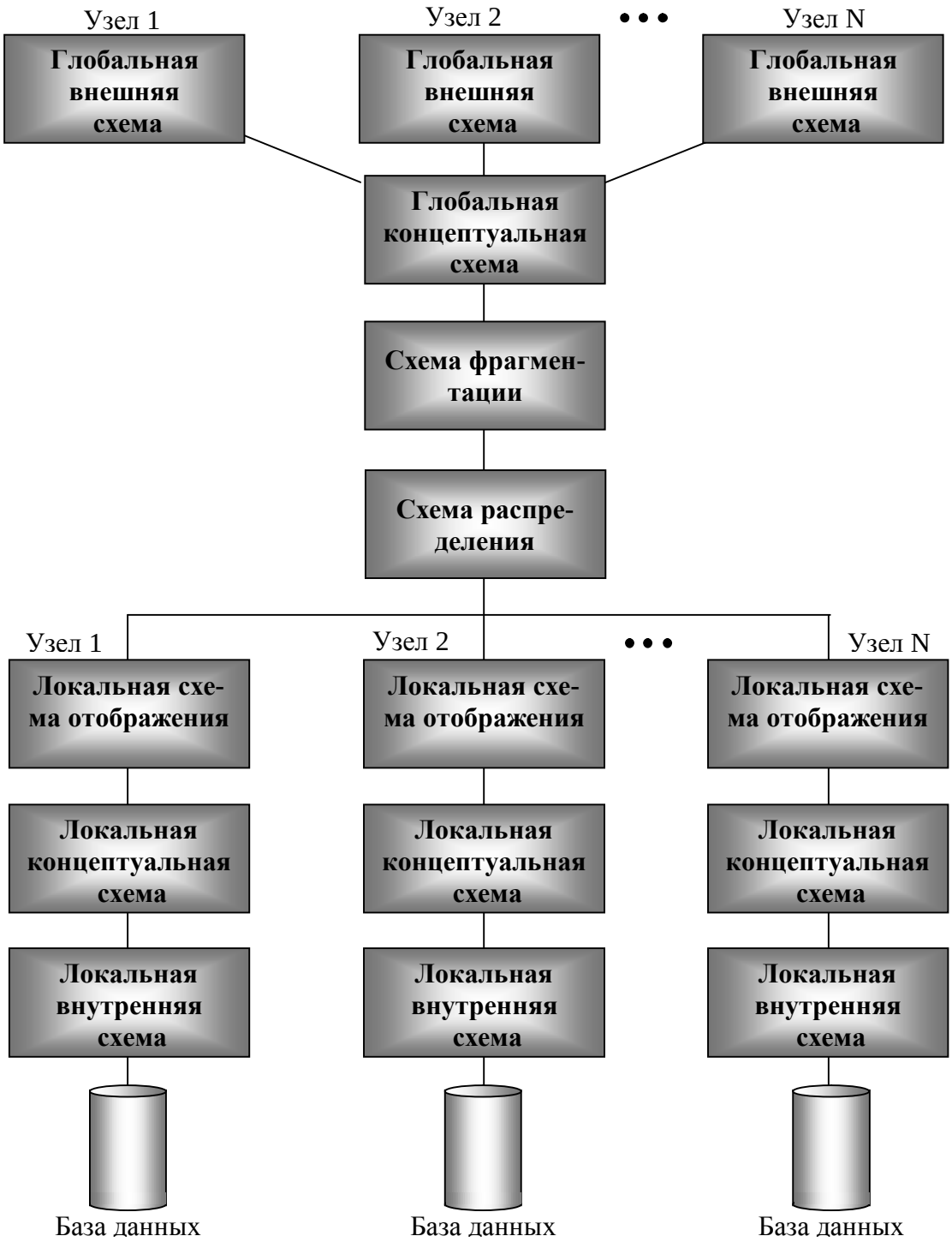


Рис. 1.4. Архитектура, рекомендуемая для РСУБД

- набор глобальных внешних схем;
- глобальную концептуальную схему;
- схему фрагментации и схему распределения;
- набор схем для каждой локальной СУБД, отвечающих требованиям трехуровневой архитектуры ANSI-SPARC.

Соединительные линии на схеме представляют преобразования, выполняемые при переходе между схемами различных типов. В зависимости от поддерживаемого уровня прозрачности некоторые из уровней рекомендуемой архитектуры могут быть опущены.

Глобальная концептуальная схема представляет собой логическое описание всей базы данных, представляющее ее так, как будто она не является распределенной. Этот уровень РСУБД соответствует концептуальному уровню архитектуры ANSI-SPARC и содержит определения сущностей, связей, требований защиты и ограничений поддержки целостности информации. Он обеспечивает физическую независимость данных от распределенной среды. Логическую независимость данных обеспечивают *глобальные внешние схемы*.

Схема фрагментации содержит описание того, как данные должны логически распределяться по разделам. *Схема распределения* является описанием того, где расположены имеющиеся данные. Схема распределения учитывает все организованные в системе процессы репликации.

Каждая локальная СУБД имеет свой собственный набор схем. *Локальная концептуальная* и *локальная внутренняя схемы* полностью соответствуют уровням архитектуры ANSI-SPARC. *Локальная схема отображения* используется для отображения фрагментов в схеме распределения во внутренние объекты локальной базы данных. Эти элементы являются зависимыми от типа используемой СУБД и служат основой для построения гетерогенных РСУБД.

Наряду с общей архитектурой РСУБД рассмотрим детализированную топологию РСУБД, которая называется *компонентной архитектурой* РСУБД [2] и должна включать четыре следующих важнейших компонента:

- ♦ локальную СУБД;
- ♦ компонент передачи данных;
- ♦ глобальный системный каталог;
- ♦ распределенную СУБД (РСУБД).

Общий вид компонентной архитектуры РСУБД с топологией, показанной на рис. 1.1, представлен на рис. 1.5. Для упрощения узлы 2 и 4 на схеме опущены, так как их структура не отличается от структуры узла 1.

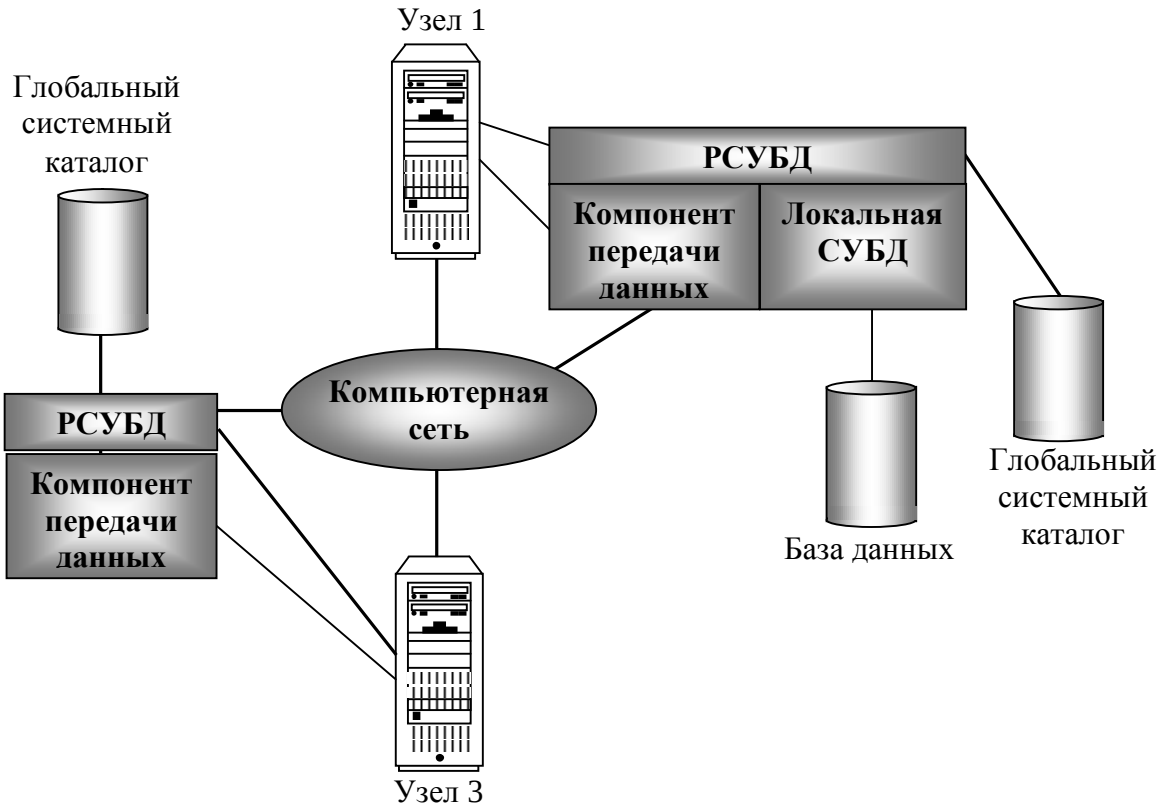


Рис. 1.5. Компонентная архитектура распределенной СУБД

Локальная СУБД – это компонент, представляющий собой стандартную СУБД для управления локальными данными на каждом узле, входящем в состав распределенной базы данных. Локальная СУБД имеет свой собственный системный каталог (называемый еще словарем данных [3]), в котором содержится информация о данных, хранящихся на этом узле. Локальные СУБД могут представлять как один и тот же программный продукт на каждом узле для гомогенных систем, так и различные программные продукты для гетерогенных систем.

Компонент передачи данных – это программное обеспечение, позволяющее всем узлам взаимодействовать между собой. Он содержит информацию об узлах и линиях связи между ними.

Глобальный системный каталог выполняет те же самые функции, что и системный каталог в централизованных базах данных. Однако глобальный каталог содержит информацию, специфическую для распределенной природы системы, например, схемы фрагментации и распределения. Этот каталог сам по себе может являться распределенной базой данных и поэтому может подвергаться фрагментации и распределению, быть

полностью реплицируемым или централизованным, как и любое другое отношение.

Компонент распределенной СУБД является управляющим по отношению ко всей системе элементом. Типичная РСУБД должна обеспечивать, по крайней мере, тот же набор функциональных возможностей, что и для централизованных СУБД. Кроме того, РСУБД должна предоставлять следующий набор функциональных возможностей [2].

- Расширенные службы установки соединений должны обеспечивать доступ к удаленным узлам и позволять передавать запросы и данные между узлами сети.
- Расширенные средства ведения каталога, позволяющие сохранять сведения о распределении данных в сети.
- Средства обработки распределенных запросов, включая механизмы оптимизации запросов и организации удаленного доступа.
- Расширенные функции управления параллельностью, позволяющие поддерживать целостность реплицируемых данных.
- Расширенные функции восстановления, учитывающие возможность отказов в работе отдельных узлов и отказов линий связи.

Вопросы

1.10. Опишите рекомендуемую архитектуру распределенной СУБД.
1.11. Опишите компонентную архитектуру распределенной СУБД.
1.12. Какие функции должна выполнять распределенная СУБД?

1.7. Проектирование распределенных реляционных баз данных

В [3] рассмотрена методология инфологического проектирования централизованных реляционных баз данных. Рассмотрим следующие дополнительные аспекты проектирования распределенных реляционных систем [2].

- **Фрагментация.** Любое отношение может быть разделено на части или *фрагменты* при организации физического хранения этого отношения. Фрагменты распределяются по различным узлам. Ес-

ли во фрагмент выделяется подмножество кортежей отношения, то он называется горизонтальным фрагментом. Фрагмент называется вертикальным, если в нем используется подмножество атрибутов отношения.

- **Распределение**. Узел для хранения фрагмента выбирается исходя из некоторой оптимальной схемы размещения фрагментов.
- **Репликация**. Одной из задач РСУБД является поддержка актуальной копии некоторого фрагмента на нескольких узлах.

Определение и размещение фрагментов должно производиться на основе анализа наиболее важных приложений, определяющих особенности использования базы данных.

При проектировании должны учитываться как качественные, так и количественные показатели будущей системы. Распределение выполняется на основе количественной информации, а базой при создании схемы фрагментации являются качественные показатели. К количественным показателям относятся следующие:

- частота запуска приложения на выполнение;
- узел, на котором запускается приложение;
- требования к производительности транзакций и приложений.

Качественная информация может включать перечень транзакций, выполняемых в приложении, используемые в этих транзакциях отношения, атрибуты и кортежи, тип доступа к этим объектам (чтение или запись), предикаты, используемые в операциях чтения.

Разделение отношений на фрагменты и распределение фрагментов по узлам выполняется для достижения следующих целей [2].

- *Локальность ссылок.* Данные должны храниться как можно ближе к местам их использования. Если фрагмент используется несколькими узлами, может оказаться целесообразным разместить на этих узлах его реплики.
- *Повышение надежности и доступности*. Надежность и доступность данных повышаются за счет использования механизма репликации. В случае отказа одного из узлов всегда будет существовать копия фрагмента, сохраняемая на другом узле.
- *Достаточный уровень производительности*. Неправильный выбор схемы размещения данных может привести к возникновению

в системе узких мест. Некоторый узел может быть перегружен запросами со стороны других узлов, что приведет к снижению производительности всей системы. С другой стороны, некоторые узлы будут "простаивать", т.е. ресурсы системы при неправильном распределении будут использоваться неэффективно.

- *Приемлемое соотношение между стоимостью и емкостью внешней памяти*. На всех узлах обычно рекомендуется использовать более дешевые устройства массовой памяти. Это требование должно быть согласовано с требованием поддержки *локальности ссылок*.
- *Минимизация расходов на передачу данных*. Стоимость выполнения в системе удаленных запросов – одна из важнейших характеристик системы. Затраты на выборку будут минимальны при обеспечении максимальной локальности ссылок, т.е. когда каждый узел будет иметь свою собственную копию данных. Однако при обновлении реплицируемых данных внесенные изменения придется распространять на все узлы с репликами, что вызовет увеличение затрат на передачу данных.

Вопросы

1.13. Одна из интенсивно развивающихся областей теории распределенных систем связана с выработкой методологии разработки распределенных баз данных. Назовите главные особенности, которые должны учитываться при проектировании распределенных баз данных. Поясните, как эти вопросы связаны с глобальным системным каталогом.

1.14. В чем состоят стратегические цели определения и распределения фрагментов?

1.7.1. Распределение данных

Существует четыре схемы размещения данных в системе [2]: централизованное, раздельное (фрагментированное), размещение с полной репликацией и размещение с выборочной репликацией. Рассмотрим эти схемы подробнее с точки зрения достижения целей, определенных выше.

Централизованное размещение

На одном из узлов под управлением СУБД создается и хранится единственная база данных. Доступ к этой базе имеют все пользователи сети (распределенная обработка, рассмотренная в 1.2.1). В этом случае локальность ссылок минимальна для всех узлов, кроме центрального, так как для получения любого доступа к данным требуется установка сетевого соединения. Соответственно уровень затрат на передачу данных будет высок. Уровень надежности и доступности в системе низок, поскольку аварийная ситуация на центральном узле приведет к отказу всей системы.

Локальность ссылок - самая низкая.
Надежность и доступность - самая низкая.
Производительность - неудовлетворительная.
Стоимость устройств хранения - самая низкая.
Затраты на передачу данных - самые высокие.

Раздельное (фрагментированное) размещение

База данных разбивается на непересекающиеся фрагменты, каждый из которых размещается на одном из узлов системы. Уровень локальных ссылок будет высок, если на узлах размещены именно те элементы данных, которые чаще всего используются на этих узлах. Если репликация не используется, то стоимость хранения данных будет минимальна, но при этом будет также невысок уровень надежности и доступности данных в системе. Однако он будет выше, чем в предыдущем варианте, поскольку аварийная ситуация на любом из узлов вызовет отказ в доступе только к той части данных, которая на нем хранилась. При правильно выбранной схеме распределения данных уровень производительности в системе будет относительно высоким, а уровень затрат на передачу данных – низким.

Локальность ссылок - высокая.
Надежность и доступность - низкая для отдельных элементов; высокая для системы в целом.
Производительность - удовлетворительная.
Стоимость устройств хранения - самая низкая.
Затраты на передачу данных - низкие.

Размещение с полной репликацией

Полная копия всей базы данных размещается на каждом из узлов системы. Следовательно, локальность ссылок, надежность и доступность данных и уровень производительности системы будут максимальны. Однако стоимость устройств хранения данных и уровень затрат на передачу данных в этом случае будут также самыми высокими.

Локальность ссылок	-	**самая высокая.**
Надежность и доступность	-	**самая высокая.**
Производительность	-	**хорошая для операций чтения.**
Стоимость устройств хранения	-	**самая высокая.**
Затраты на передачу данных	-	**высокие для операций обновления, низкие для операций чтения.**

Размещение с выборочной репликацией

Эта схема представляет собой комбинацию методов фрагментации, репликации и централизации. Одни массивы данных разделяются на фрагменты, что позволяет добиться для них высокой локальности ссылок. Другие данные, используемые на многих узлах, но не часто обновляемые, реплицируются. Остальные данные хранятся централизованно. Такая стратегия позволяет объединить все преимущества, существующие в остальных моделях, и исключить свойственные им недостатки. Благодаря своей гибкости именно эта стратегия используется чаще всего.

Локальность ссылок	-	**высокая.**
Надежность и доступность	-	**низкая для отдельных элементов; высокая для системы в целом.**
Производительность	-	**удовлетворительная.**
Стоимость устройств хранения	-	**средняя.**
Затраты на передачу данных	-	**низкие.**

Вопросы

1.15. Сравните способы размещения данных в системе по уровню локальности ссылок.

1.16. Сравните способы размещения данных в системе по уровню надежности и доступности.

1.17. Сравните способы размещения данных в системе по уровню производительности.

1.18. Сравните способы размещения данных в системе по уровню стоимости устройств хранения.

1.19. Сравните способы размещения данных в системе по уровню затрат на передачу данных.

1.7.2. Фрагментация

Система поддерживает **фрагментацию**, если данное отношение может быть разделено на части или *фрагменты* при организации его физического хранения. Фрагментация желательна для повышения эффективности системы. В этом случае данные могут храниться в том месте, где они чаще всего используются. Это позволяет достичь локализации большинства операций и уменьшения сетевого трафика. Кроме того, исключается хранение данных, которые не используются локальными приложениями, а значит, неавторизированные пользователи не смогут получить к ним доступ.

С другой стороны, производительность приложений, требующих доступа к данным из нескольких фрагментов на различных узлах, может оказаться недостаточной. Поддержка целостности данных также может существенно осложняться, так как функционально зависимые данные могут оказаться фрагментированными и размещаться на различных узлах.

Для корректности фрагментации необходимо выполнение следующих правил [2].

1. *Полнота*. Каждый элемент данных из исходного отношения должен присутствовать, по крайней мере, в одном из созданных фрагментов. Это гарантирует отсутствие потери информации при фрагментации.

2. *Восстановимость*. Исходное отношение должно быть восстановимо из его фрагментов при помощи операций реляционной алгебры. Это гарантирует сохранение функциональных зависимостей.
3. *Непересекаемость*. Один элемент данных не должен присутствовать в двух и более фрагментах. Исключение составляет вертикальная фрагментация, когда в каждом фрагменте должны присутствовать атрибуты первичного ключа, необходимые для восстановления исходного отношения. Это правило гарантирует минимальную избыточность данных во фрагментах.

Существуют два основных типа фрагментации: **горизонтальная** и **вертикальная**. Горизонтальные фрагменты представляют собой подмножества кортежей отношения, а вертикальные – подмножества атрибутов отношения, как показано на рис. 1.6.

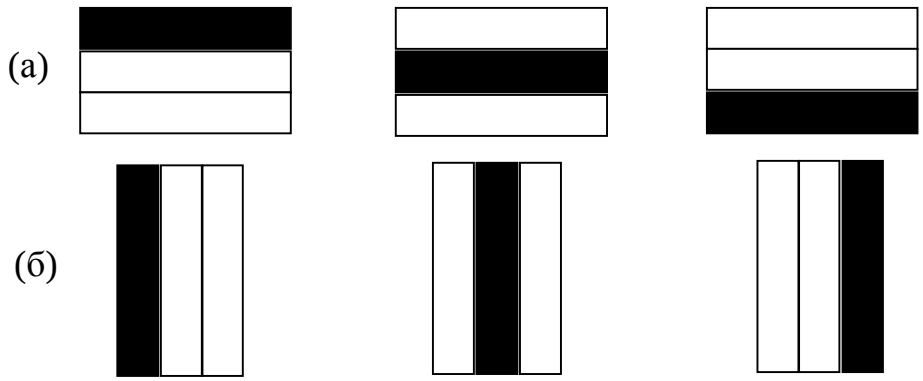


Рис. 1.6. Различные типы фрагментации:
а) – горизонтальная; б) – вертикальная

Горизонтальная фрагментация

Горизонтальным называется фрагмент, выделенный из отношения по горизонтали и состоящий из некоторого подмножества кортежей этого отношения.

Для создания горизонтального фрагмента определяется предикат, с помощью которого выполняется отбор кортежей из исходного отношения. Этот тип фрагмента определяется с помощью операции *выборки* (*селекции*) реляционной алгебры. Операция выборки позволяет отобрать группу кортежей, имеющих некоторое общее для них свойство, – например, все кортежи, используемые одним из приложений, или все кортежи,

применяемые на одном из узлов. Если задано отношение R_1, то его горизонтальный фрагмент может быть определен формулой [4]:

$$R = \sigma_F(R_1)$$

Здесь F является предикатом, построенным с использованием одного или больше атрибутов отношения R_1.

Пример 1.6.1. Пусть информация о служащих имеет вид отношения СЛУЖ (Табл. 1.1). Оно содержит атрибуты: табельный номер работника (Таб#), номер отделения компании (Отд#), в котором он работает, фамилия, имя, отчество (Фио), пол (Пол), дата рождения (Рожд), номер паспорта (Пасп#), должность (Долж), оклад (Оклад).

Таблица 1.1

Отношение СЛУЖ

Таб#	Отд#	Фио	Пол	Рожд	Пасп#	Долж	Оклад
154	Д1	Иванов И.И.	М	10.11.60	153238	ассистент	3200
155	Д1	Петров П.П.	М	13.04.51	473850	менеджер	4000
156	Д2	Зайцева С.П.	Ж	11.05.69	872134	ассистент	2500
157	Д2	Сидоров С.С.	М	15.12.70	876234	менеджер	3000
158	Д3	Медведева О.П.	Ж	23.03.68	786349	менеджер	3500

Пусть известно, что отделение Д1 расположено в Москве, а отделения Д2 и Д3 – в Воронеже. В этом случае горизонтальная фрагментация отношения СЛУЖ по атрибуту Отд# может быть выполнена следующим образом:

$$М\_СЛУЖ = \sigma_{Отд\#='Д1'}(СЛУЖ)$$

$$В\_СЛУЖ = \sigma_{Отд\#='Д2' \vee Отд\#='Д3'}(СЛУЖ)$$

В результате будут созданы два фрагмента. Первый, содержимое которого представлено в табл. 1.2, будет состоять из кортежей, в которых значение атрибута Отд# будет равно 'Д1'. Этот фрагмент имеет внутрисистемное имя М_СЛУЖ и будет храниться на узле в Москве. Второй фрагмент с именем В_СЛУЖ (табл. 1.3) будет храниться на узле в Воронеже и состоять из кортежей, в которых значение атрибута Отд# равно 'Д2'.

Таблица 1.2

Горизонтальный фрагмент М_СЛУЖ отношения СЛУЖ

Таб#	Отд#	Фио	Пол	Рожд	Пасп#	Должс	Оклад
154	Д1	Иванов И.И.	М	10.11.60	153238	ассистент	3200
155	Д1	Петров П.П.	М	13.04.51	473850	менеджер	4000

Таблица 1.3

Горизонтальный фрагмент В_СЛУЖ отношения СЛУЖ

Таб#	Отд#	Фио	Пол	Рожд	Пасп#	Должс	Оклад
156	Д2	Зайцева С.П.	Ж	11.05.69	872134	ассистент	2500
157	Д2	Сидоров С.С.	М	15.12.70	876234	менеджер	3000
158	Д3	Медведева О.П.	Ж	23.03.68	786349	менеджер	3500

Предложенная схема фрагментации отвечает всем правилам корректности.

- *Полнота*. Каждый кортеж исходного отношения присутствует либо во фрагменте М_СЛУЖ, либо во фрагменте В_СЛУЖ.
- *Восстановимость*. Отношение СЛУЖ может быть восстановлено из созданных фрагментов с помощью следующей операции объединения:

$$\text{СЛУЖ} = \text{М\_СЛУЖ} \cup \text{В\_СЛУЖ}$$

- *Непересекаемость*. Полученные фрагменты не пересекаются, поскольку не существует значения атрибута Отд# , которое одновременно было бы равно значениям 'Д1' и 'Д2' или 'Д3'.

В одних случаях целесообразность использования горизонтальной фрагментации очевидна. Когда же это не так, потребуется выполнение детального анализа приложений. Анализ должен включать проверку предикатов поиска, используемых в транзакциях или запросах, выполняемых в приложении. Предикаты могут быть простыми, включающими только по одному атрибуту, или сложными, включающими несколько атрибутов. Для каждого из используемых атрибутов предикат может содержать единственное значение или несколько значений. В последнем случае значения могут быть дискретными или представлять диапазон значений.

Построение схемы фрагментации предполагает поиск набора **минимальных** (т.е. полных и релевантных) предикатов для разбиения от-

ношения на фрагменты [2]. Набор предикатов является **полным** тогда и только тогда, когда вероятность обращения к любым двум кортежам одного и того же фрагмента со стороны любого приложения будет одинакова. Предикат является **релевантным**, если существует, по крайней мере, одно приложение, которое по-разному обращается к выделенным с помощью этого предиката фрагментам.

Вертикальная фрагментация

Вертикальным называется фрагмент, выделенный из отношения по вертикали и состоящий из подмножества атрибутов этого отношения.

При вертикальной фрагментации в различные фрагменты объединяются атрибуты, используемые отдельными приложениями. Определение фрагментов в этом случае выполняется с помощью операции проекции реляционной алгебры [4]. Для заданного отношения R_1 вертикальный фрагмент может быть вычислен с помощью следующей формулы:

$$R = \pi_{a1,\mathbf{K},an}(R_1)$$

Здесь a1,…,an представляют собой атрибуты отношения R_1.

Пример 1.6.2. В качестве исходного будем использовать отношение СЛУЖ (табл. 1.1), рассмотренное выше. Приложение, печатающее платежные ведомости, для каждого из работников компании использует атрибуты табельного номера работника (Таб#), должность (Долж), пол (Пол), дата рождения (Рожд), номер паспорта (Пасп#), оклад (Оклад). Ведомость, выдаваемая для отдела кадров, содержит атрибуты Таб# (табельный номер), Фио (Фамилия, имя, отчество) и Отд# (номер отделения компании). Исходя из этих сведений, вертикальная фрагментация отношения СЛУЖ может быть выполнена с помощью следующих определений:

$$С_1 = \pi_{Таб\#,Долж,Пол,Рожд,Пасп\#,Оклад}(СЛУЖ)$$

$$С_2 = \pi_{Таб\#,Фио,Отд\#}(СЛУЖ)$$

С помощью этих формул будут созданы два фрагмента, содержимое которых представлено в табл. 1.4 и 1.5. При этом оба фрагмента содержат первичный ключ – атрибут Таб# – что позволяет при необходимости реконструировать исходное отношение. Преимущество вертикальной фрагментации состоит в том, что отдельные фрагменты могут раз-

мещаться на тех сайтах, на которых они используются. Это дополнительно оказывает положительное влияние на производительность системы, поскольку размеры каждого из фрагментов меньше размеров исходной таблицы.

Таблица 1.4

Вертикальный фрагмент C_1 отношения СЛУЖ

Таб#	Пол	Рожд	Пасп#	Долж	Оклад
154	М	10.11.60	153238	ассистент	3200
155	М	13.04.51	473850	менеджер	4000
156	Ж	11.05.69	872134	ассистент	2500
157	М	15.12.70	876234	менеджер	3000
158	Ж	23.03.68	786349	менеджер	3500

Таблица 1.5

Вертикальный фрагмент C_2 отношения СЛУЖ

Таб#	Отд#	Фио
154	Д1	Иванов И.И.
155	Д1	Петров П.П.
156	Д2	Зайцева С.П.
157	Д2	Сидоров С.С.
158	Д3	Медведева О.П.

Приведенная схема фрагментации удовлетворяет правилам корректности.

- *Полнота*. Каждый атрибут отношения СЛУЖ присутствует либо во фрагменте C_1, либо во фрагменте C_2.
- *Восстановимость*. Исходное отношение СЛУЖ может быть реконструировано из отдельных фрагментов с помощью операции естественного соединения:

$$\text{СЛУЖ} = C_1 >< C_2$$

- *Непересекаемость*. Содержимое отдельных фрагментов не пересекается, за исключением атрибута первичного ключа Таб#, необходимого для реконструкции исходного отношения.

Смешанная фрагментация

Иногда для адекватного распределения данных между приложениями только горизонтальной или только вертикальной фрагментации оказывается недостаточно. В таких случаях используют **смешанную** фрагментацию.

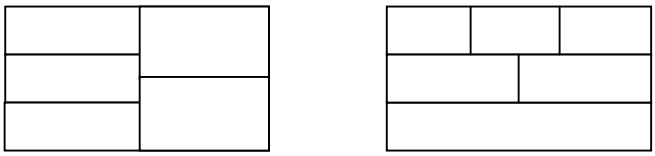
Смешанный фрагмент образуется либо после дополнительной вертикальной фрагментации созданных ранее горизонтальных фрагментов, либо путем горизонтальной фрагментации определенных ранее вертикальных фрагментов (рис 1.7).

Смешанная фрагментация определяется сочетанием операций выборки и проекции реляционной алгебры. Для существующего отношения R_1 смешанный фрагмент можно определить по формулам:

$$R = \sigma_F(\pi_{a1,\mathbf{K},an}(R_1)) \text{ или}$$

$$R = \pi_{a1,\mathbf{K}an}(\sigma_F(R_1))$$

Здесь F – предикат, построенный с использованием одного или более атрибутов a1,…an отношения R_1.



(а) (б)

Рис. 1.7. Смешанная фрагментация:
а) – горизонтально разделенные вертикальные фрагменты;
б) – вертикально разделенные горизонтальные фрагменты

Пример 1.6.3. Выше отношение СЛУЖ было разбито на два вертикальных фрагмента (табл. 1.4, 1.5) для приложений печати платежной ведомости и некоторого документа отдела кадров. Для разбиения использовались формулы:

$$C_1 = \pi_{Таб\#,Долж,Пол,Рожд,Пасп\#,Оклад}(СЛУЖ)$$

$$C_2 = \pi_{Таб\#,Фио,Отд\#}(СЛУЖ)$$

Теперь можно выполнить дополнительную горизонтальную фрагментацию фрагмента C_2 по атрибуту номера отделения компании Таб#:

$$C_{21} = \sigma_{Отд\#='Д1'}(C_2)$$

$$C_{22} = \sigma_{Отд\#='Д2'}(C_2)$$

$$C_{23} = \sigma_{Отд\#='Д3'}(C_2)$$

В результате будут созданы три фрагмента, первый из которых включает кортежи с номером отделения, равным 'Д1' (табл. 1.7), второй – с номером отделения, равным 'Д2' (табл. 1.8), а в третий вошли кортежи с номером отделения, равным 'Д3' (табл. 1.9). Содержимое фрагмента C_1 показано в табл. 1.6

Таблица 1.6

Смешанная фрагментация отношения СЛУЖ. Фрагмент C_1

Таб#	Пол	Рожд	Пасп#	Долж	Оклад
154	М	10.11.60	153238	ассистент	3200
155	М	13.04.51	473850	менеджер	4000
156	Ж	11.05.69	872134	ассистент	2500
157	М	15.12.70	876234	менеджер	3000
158	Ж	23.03.68	786349	менеджер	3500

Таблица 1.7

Смешанная фрагментация отношения СЛУЖ. Фрагмент C_{21}

Таб#	Отд#	Фио
154	Д1	Иванов И.И.
155	Д1	Петров П.П.

Таблица 1.8

Смешанная фрагментация отношения СЛУЖ. Фрагмент C_{22}

Таб#	Отд#	Фио
156	Д2	Зайцева С.П.
157	Д2	Сидоров С.С.

Таблица 1.9

Смешанная фрагментация отношения СЛУЖ. Фрагмент C_{23}

Таб#	Отд#	Фио
158	Д3	Медведева О.П.

Полученная схема фрагментации удовлетворяет правилам корректности.

- *Полнота*. Каждый из атрибутов исходного отношения СЛУЖ присутствует либо во фрагменте C_1, либо во фрагменте C_2; каждый

кортеж (в виде отдельных частей) исходного отношения присутствует во фрагменте C_1, а также в отношении C_{21}, C_{22} или C_{23}.

- *Восстановимость*. Исходное отношение СЛУЖ может быть восстановлено из полученных фрагментов путем выполнения операций объединения и естественного соединения по следующей формуле:

$$СЛУЖ = C_1 >< (C_{21} \cup C_{22} \cup C_{23})$$

- *Непересекаемость*. Полученные фрагменты не пересекаются, так как фрагменты C_1 и C_2 содержат различные атрибуты (за исключением обязательного атрибута первичного ключа) и не существует работника, нанятого более чем в одно отделение компании.

Если отношение содержит небольшое количество кортежей, которые относительно редко обновляются, то от фрагментации **отказываются**. Такое отношение оставляют нефрагментированным и просто размещают на каждом из узлов его реплицируемые копии.

Поиск отношений, которые не нуждаются во фрагментации – первый этап процедуры определения схемы фрагментации. Затем анализируют отношения, расположенные на единичной стороне связей типа "один ко многим", и подбирают для них оптимальные схемы фрагментации. На последнем этапе анализируются отношения, расположенные на множественной стороне тех же связей.

Вопросы

1.20. Дайте определение и укажите различия между альтернативными схемами фрагментации глобальных отношений.

1.21. Поясните, как можно проверить корректность выполненных действий при фрагментации и получить гарантии того, что в процессе фрагментации в базу данных не было внесено семантических изменений.

1.7.3. Репликация

Репликацию можно определить как процесс генерации и воспроизведения нескольких копий данных, размещаемых на одном или нескольких узлах.

Механизм репликации очень важен, поскольку позволяет организации обеспечивать доступ пользователям к актуальным данным там и тогда, когда они в этом нуждаются. Использование репликации позволяет достичь многих преимуществ, включая повышение производительности (в тех случаях, когда централизованный ресурс оказывается перегруженным), надежности хранения и доступности данных, наличие «горячей» резервной копии на случай восстановления, а также возможность поддержки мобильных пользователей и хранилищ данных.

Виды репликации

Протоколы обновления реплицируемых данных построены на допущении, что обновления всех копий данных выполняются как часть самой транзакции обновления. Другими словами, все копии реплицируемых данных обновляются одновременно с изменением исходной копии и, как правило, с помощью протокола двухфазной фиксации транзакций [2]. Такой вариант репликации называется *синхронной репликацией*.

Хотя этот механизм может быть просто необходим для некоторого класса систем, в которых все копии данных требуется поддерживать в абсолютно синхронном состоянии (например, в случае финансовых операций), ему свойственны определенные недостатки. В частности, транзакция не сможет быть завершена, если один из узлов с копией реплицируемых данных окажется недоступным. Кроме того, множество сообщений, необходимых для координации процесса синхронизации данных, создают существенную дополнительную нагрузку на корпоративную сеть.

Многие коммерческие распределенные СУБД предоставляют другой механизм репликации, получивший название *асинхронной репликации*. Он предусматривает обновление целевых баз данных после выполнения обновления исходной базы данных. Задержка в восстановлении согласованности данных может варьироваться от нескольких секунд до нескольких часов или даже дней. Однако рано или поздно данные во всех копиях будут приведены в синхронное состояние. Хотя такой подход нарушает принцип независимости распределенных данных, он вполне может пониматься как приемлемый компромисс между целостностью данных и их доступностью, причем последнее может быть важнее для орга-

низаций, чья деятельность допускает работу с копией данных, необязательно точно синхронизованной на текущий момент.

Функции службы репликации

В качестве базового уровня служба репликации распределенных данных должна копировать данные из одной базы данных в другую синхронно или асинхронно. Кроме того, существует множество других функций, которые должны поддерживаться, включая следующие [2].

- *Масштабируемость*. Служба репликации должна эффективно обрабатывать как малые, так и большие объемы данных.
- *Отображение и трансформация*. Служба репликации должна поддерживать репликацию данных в гетерогенных системах, использующих несколько платформ. Это может быть связано с необходимостью отображения и преобразования данных из одной модели данных в другую или же для преобразования некоторого типа данных в соответствующий тип данных, но для среды другой СУБД.
- *Репликация объектов*. Должна существовать возможность реплицировать объекты, отличные от обычных данных. Например, в некоторых системах допускается репликация индексов и хранимых процедур (или триггеров).
- *Средства определения схемы репликации*. Система должна предоставлять механизм, позволяющий привилегированным пользователям задавать данные и объекты, подлежащие репликации.
- *Механизм подписки*. Система должна включать механизм, позволяющий привилегированным пользователям оформлять подписку на данные и другие подлежащие репликации объекты.
- Механизм инициализации. Система должна включать средства, обеспечивающие инициализацию вновь создаваемой реплики.

Схемы владения данными

Владение данными определяет, какому из узлов будет предоставлена привилегия обновления данных. Основными типами схем владения являются [2]:

- «ведущий/ведомый»;

- «рабочий поток»;
- «повсеместное обновление».

Последний вариант иногда называют *одноранговой*, или *симметричной репликацией*.

При организации владения данными по схеме «ведущий/ведомый» асинхронно реплицируемые данные принадлежат одному из узлов, называемому ведущим, или первичным, и могут обновляться только на нем. Здесь можно провести аналогию между издателем и подписчиками. Издатель (ведущий узел) публикует свои данные. Все остальные узлы только лишь подписываются на данные, принадлежащие ведущему сайту, т.е. имеют собственные локальные копии, доступные им только для чтения. Потенциально каждый из сайтов может играть роль ведущего для различных, не перекрывающихся наборов данных. Однако в системе может существовать только один узел, на котором располагается ведущая обновляемая копия каждого конкретного набора данных, а это означает, что конфликты обновления данных в системе полностью исключены. Ниже приводится несколько примеров возможных вариантов использования этой схемы репликации.

- *Системы, поддержки принятия решений* (ППР). Данные из одной или более распределенных баз данных могут выгружаться в отдельную, локальную систему ППР, где они будут только считываться при выполнении различных видов анализа.
- *Централизованное распределение или распространение информации*. Распространение данных имеет место в тех случаях, когда данные обновляются только в центральном звене системы, после чего реплицируются их копии, доступные только для чтения. Этот вариант репликации данных показан на рис.1.8, а.
- *Консолидация удаленной информации*. Консолидация данных имеет место в тех случаях, когда обновление данных выполняется локально, поле чего их копии, доступные только для чтения, отсылаются в общее хранилище. В этой схеме каждый из сайтов автономно владеет некоторой частью данных. Этот вариант репликации данных показан на рис. 1.8, б.

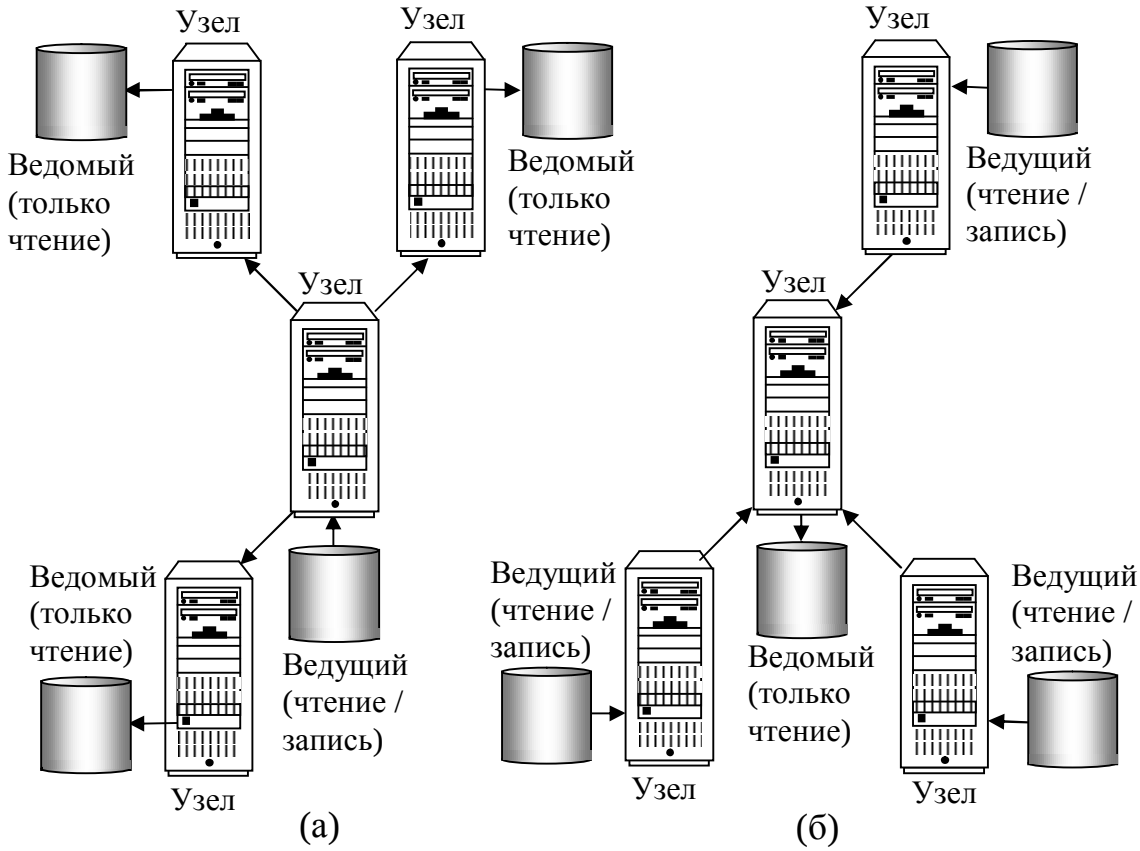


Рис. 1.8. Владение данными по схеме «ведущий/ведомый»: а) распределение данных; б)консолидация данных

- *Поддержка мобильных пользователей*. Поддержка работы мобильных пользователей получила в последние годы очень широкое распространение. Сотрудники многих организаций вынуждены постоянно перемещаться с места на место и работать за пределами офисов. Разработано несколько методов предоставления необходимых данных мобильным пользователям. Одним из них и является репликация. В этом случае по требованию пользователя данные загружаются с локального сервера его рабочей группы. Обновления, выполненные клиентом для данных рабочей группы или центрального сайта, обрабатываются сходным образом.

Ведущий сайт может владеть данными всей таблицы, и в этом случае все остальные сайты являются лишь подписчиками на копии этой таблицы, доступные только для чтения. В альтернативном варианте многие сайты владеют отдельными фрагментами таблицы, а остальные сайты могут выступать как подписчики копий каждого из этих фрагментов,

доступных им только для чтения. Этот тип репликации называют *асимметричной репликацией*.

Как и в случае схемы «ведущий/ведомый», в модели «*рабочий поток*» удается избежать появления конфликтов обновления, хотя данной модели свойствен больший динамизм. Схема владения «рабочий поток» позволяет передавать право обновления реплицируемых данных от одного сайта другому. Однако в каждый конкретный момент времени существует только один сайт, имеющий право обновлять некоторый конкретный набор данных. Типичным примером использования схемы рабочего потока является система обработки заказов, в которой работа с каждым заказом выполняется в несколько этапов, например оформление заказа, контроль кредитоспособности, выписка счета, доставка и т.д.

Централизованные системы позволяют приложениям, выполняющим отдельные этапы обработки, получать доступ и обновлять данные в одной интегрированной базе данных. Каждое приложение обновляет данные о заказе по очереди тогда и только тогда, когда состояние заказа указывает, что предыдущий этап обработки уже завершен. В модели владения «рабочий поток» приложения могут быть распределены по различным сайтам, и когда данные реплицируются и пересылаются на следующий сайт в цепочке, вместе с ними передается и право на их обновление, как показано на рис. 1.9.

У двух предыдущих моделей есть одно общее свойство: в любой заданный момент времени только один узел имеет право обновлять данные. Всем остальным сайтам доступ к репликам данным будет разрешен только для чтения. В некоторых случаях это ограничение оказывается слишком жестким.

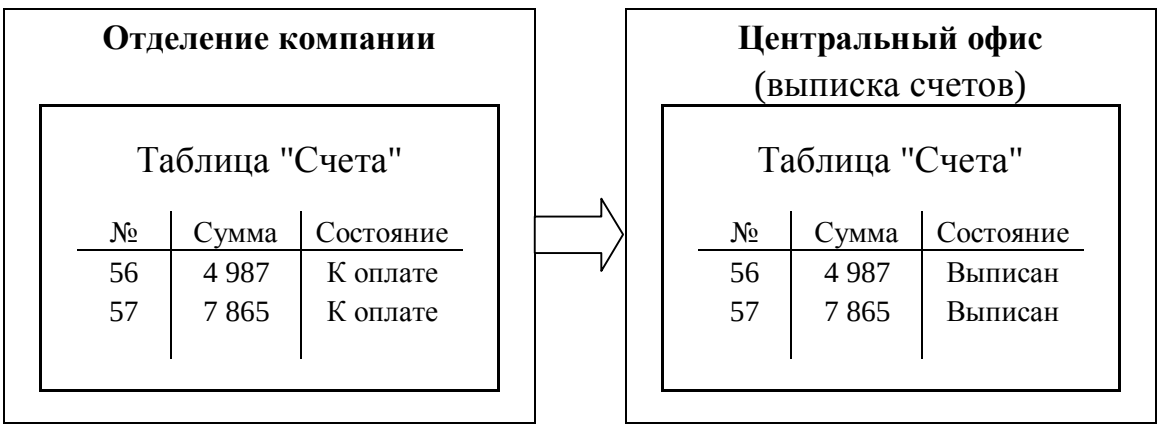


Рис. 1.9. Владение данными по схеме «рабочий поток»

Схема владения *с повсеместным обновлением* создает равноправную среду, в которой множество узлов имеют одинаковые права на обновление реплицируемых данных. В результате локальные узлы полу чают возможность работать автономно, даже в тех случаях, когда другие сайты недоступны.

Разделение права владения может вызвать возникновение в системе конфликтов, поэтому служба репликации в этой схеме должна использовать тот или иной метод выявления и разрешения конфликтов.

Сохранение целостности транзакций

Первые попытки реализации механизма репликации по самой своей сути не предусматривали сохранения целостности транзакций [2]. Данные копировались без сохранения свойства атомарности транзакций, что потенциально могло привести к утрате целостности распределенных данных. Напомним, что атомарность транзакции означает, что выполняются все операции, входящие в нее, или ничего.

В данном случае транзакция, состоящая на исходном сайте из нескольких операций обновления данных в различных таблицах, в процессе репликации преобразовывалась в серию отдельных транзакций, каждая из которых обновляла данные в одной из таблиц. Если часть этих транзакций на целевом сайте завершалась успешно, а остальная часть – нет, то согласованность информации между двумя базами данных утрачивалась.

В противоположность этому, репликация с сохранением структуры транзакции в исходной базе данных не нарушает свойство атомарности транзакции.

Моментальные снимки таблиц

Метод *моментальных снимков таблиц* позволяет асинхронно распространять результаты изменений, выполненных в отдельных таблицах, группах таблиц, представлениях или фрагментах таблиц в соответствии с заранее установленным графиком [2].

Общий подход к созданию моментальных снимков состоит в использовании файла журнала восстановления базы данных, который позволяет минимизировать уровень дополнительной нагрузки на систему. Основная идея состоит в том, что файл журнала является лучшим источ-

ником для получения сведений об изменениях в исходных данных. Достаточно иметь механизм, который будет обращаться к файлу журнала для выявления изменений в исходных данных, после чего распространять обнаруженные изменения на целевые базы данных, не оказывая никакого влияния на нормальное функционирование исходной системы. Различные СУБД реализуют подобный механизм по-разному: в некоторых случаях данный процесс является частью самого сервера СУБД, тогда как в других случаях он реализуется как независимый внешний сервер.

Для отправки сведений об изменениях на другие сайты целесообразно применять метод организации очередей. Если произойдет отказ сетевого соединения или целевого сайта, сведения об изменениях могут сохраняться в очередях до тех пор, пока соединение не будет восстановлено. Для гарантии сохранения согласованности данных порядок доставки сведений на целевые сайты должен сохранять исходную очередность внесения изменений.

Триггеры базы данных

Альтернативный подход заключается в предоставлении пользователям возможности создавать собственные приложения, выполняющие репликацию данных с использованием механизма *триггеров базы данных*.

В этом случае на пользователей возлагается ответственность за написание тех триггерных процедур, которые будут вызываться при возникновении соответствующих событий, например при создании новых записей или обновлении уже существующих. Хотя подобный подход предоставляет большую гибкость, чем механизм создания моментального снимка, ему также присущи определенные недостатки.

- Отслеживание запуска и выполнение триггерных процедур создает дополнительную нагрузку на систему.
- Триггеры выполняются при каждом изменении строки в ведущей таблице. Если ведущая таблица подвержена частым обновлениям, вызов триггерных процедур может создать существенную дополнительную нагрузку на приложения и сетевые соединения. В противоположность этому, при использовании моментальных снимков все выполненные изменения пересылаются за одну операцию.

- Триггеры не могут выполняться в соответствии с некоторым графиком. Они выполняются в тот момент, когда происходит обновление данных в ведущей таблице. Моментальные снимки могут создаваться в соответствии с установленным графиком или даже вручную. В любом случае это позволяет исключить дополнительную нагрузку от репликации данных в периоды пиковой нагрузки на систему.
- Если реплицируется несколько связанных таблиц, синхронизация их репликации может быть достигнута за счет использования механизма групповых обновлений. Решить эту задачу с помощью триггеров существенно сложнее.
- Аннулирование результатов выполнения триггерной процедуры в случае отмены или отката транзакции – достаточно сложная задача.

Выявление и разрешение конфликтов

Когда несколько узлов могут независимо вносить изменения в реплицируемые данные, необходимо использовать некоторый механизм, позволяющий выявлять конфликтующие обновления данных и восстанавливать согласованность информации в базе. Простейший механизм обнаружения конфликтов в отдельной таблице состоит в рассылке исходным сайтом как новых, так и исходных значений измененных данных для каждой строки, которая была обновлена с момента последней синхронизации копий. На целевом сайте сервер репликации должен сравнить с полученными значениями каждую строку в целевой базе данных, которая была локально изменена за данный период. Однако этот метод требует установки дополнительных соглашений для обнаружения других типов конфликтов, например нарушения ссылочной целостности между двумя таблицами.

Было предложено несколько различных механизмов разрешения конфликтов, однако чаще всего применяются следующие [2].

- *Самая ранняя или самая поздняя временная отметка*. Изменяются соответственно данные с самой ранней или самой поздней временной отметкой.
- *Приоритеты узлов*. Применяется обновление, поступившее с узла с наибольшим приоритетом.

- *Дополняющие и усредненные обновления*. Введенные изменения обобщаются. Этот вариант разрешения конфликтов может использоваться в тех случаях, когда обновление атрибута выполняется операциями, записанными в форме отклонений.
- *Минимальное или максимальное значение*. Применяются обновления, соответствующие столбцу с минимальным или максимальным значением.
- *По решению пользователя*. Администратор базы данных создает собственную процедуру разрешения конфликта. Для устранения различных типов конфликтов могут быть подготовлены различные процедуры.
- *Сохранение информации для принятия решения вручную*. Сведения о конфликте записываются в журнал ошибок для последующего анализа и устранения администратором базы данных вручную.

Вопросы

1.22. В чем различия, достоинства и недостатки синхронной и асинхронной репликации?

1.23. Какие функции должна поддерживать служба репликации?

1.24. Приведите примеры использования схемы владения данными «ведущий/ведомый».

1.25. Опишите отличия в работе централизованной системы и распределенной системы с моделью владения данными «рабочий поток» при обработке заказа.

1.26. Нарушение какого свойства транзакций при репликации приводит к утрате согласованности информации между двумя базами данных?

1.27. В чем состоит общий подход к реализации метода моментальных снимков таблиц?

1.28. Перечислите недостатки механизма триггеров базы данных.

1.29. Опишите наиболее часто применяемые механизмы разрешения конфликтов.

1.8. Обеспечение прозрачности в РСУБД

В определении РСУБД, приведенном в разделе 1.2, утверждается, что система должна обеспечить прозрачность распределенного хранения данных для конечного пользователя. Под прозрачностью понимается сокрытие от пользователей деталей реализации системы. Другими словами, пользователи распределенной системы должны иметь возможность действовать так, как если бы система *не* была распределена. Все проблемы распределенных систем должны относиться к внутренним проблемам (проблемам реализации), а не к внешним проблемам (проблемам пользовательского уровня). В централизованной СУБД обеспечение независимости программ от данных также можно рассматривать как одну из форм прозрачности – от пользователя скрываются изменения, происходящие в определении и организации хранения данных. Распределенные СУБД могут обеспечивать прозрачность на различных уровнях. При этом преследуется одна цель: сделать работу с распределенной базой данных совершенно аналогичной работе с обычной централизованной СУБД. Выделяют четыре основных типа прозрачности для системы с распределенной базой данных [2].

- Прозрачность распределенности.
- Прозрачность транзакций.
- Прозрачность выполнения.
- Прозрачность использования СУБД.

1.8.1. Прозрачность распределенности

Прозрачность распределенности базы данных позволяет конечным пользователям работать с базой данных точно так, по крайней мере, с логической точки зрения, как если бы данные в действительности были вовсе не фрагментированы. Если РСУБД обеспечивает прозрачность распределенности, то пользователю не требуется каких-либо знаний о фрагментации данных **(прозрачность фрагментации)** или их размещении **(прозрачность расположения).** Прозрачность фрагментации (как и прозрачность расположения) – это достаточно важное свойство, так как она

позволяет упростить разработку пользовательских программ и выполнение терминальных операций. Например, это гарантирует, что в любой момент данные могут быть заново восстановлены (а фрагменты перераспределены) в ответ на изменение требований к эффективности работы системы, причем ни пользовательские программы, ни терминальные операции при этом не затрагиваются.

С другой стороны, если пользователю необходимо иметь сведения о фрагментации данных и расположении фрагментов, то этот тип прозрачности называют **прозрачностью локального отображения.** Далее мы рассмотрим все упомянутые типы прозрачности. Для иллюстрации обсуждаемых концепций будет использоваться отношение СЛУЖ, фрагментированное так, как описано в примере 1.6.3 , а именно:

$C_1 = \pi_{Таб\#,Долж,Пол,Рожд,Пасп\#,Оклад}(СЛУЖ)$ фрагмент расположен на сайте 1

$C_2 = \pi_{Таб\#,Фио,Отд\#}(СЛУЖ)$

$C_{21} = \sigma_{Отд\#='Д1'}(C_2)$ фрагмент расположен на сайте 1

$C_{22} = \sigma_{Отд\#='Д2'}(C_2)$ фрагмент расположен на сайте 2

$C_{23} = \sigma_{Отд\#='Д3'}(C_2)$ фрагмент расположен на сайте 3

Прозрачность фрагментации

Прозрачность фрагментации (или независимость от фрагментации) является самым верхним уровнем прозрачности распределенности. Если РСУБД обеспечивает прозрачность фрагментации, то пользователю не требуется знать, как именно фрагментированы данные. В этом случае доступ к данным осуществляется на основе глобальной схемы и пользователю нет необходимости указывать имена фрагментов или расположение данных. Например, для выборки сведений обо всех руководителях отделений (у них атрибут Долж имеет значение 'менеджер'), при наличии в системе прозрачности фрагментации, можно пользоваться следующим SQL-оператором [3]:

```
SELECT Фио
FROM СЛУЖ
WHERE Долж='менеджер';
```

Это тот же самый SQL-оператор, который потребовалось бы ввести для получения указанных результатов в централизованной системе.

Таким образом, если обеспечивается прозрачность фрагментации, пользователи получают данные в виде некоторого представления, в котором фрагменты логически скомбинированы с помощью соответствующих операций соединения и объединения. К обязанностям *системного оптимизатора* относится определение фрагментов, к которым требуется физический доступ для выполнения любого из поступивших запросов пользователя.

Прозрачность расположения

Прозрачность расположения представляет собой средний уровень прозрачности распределенности. В этом случае пользователь должен иметь сведения о способах фрагментации данных в системе, но не нуждается в сведениях о расположении данных. При наличии в системе прозрачности расположения тот же самый запрос следует переписать в таком виде:

```
SELECT Фио
FROM С21
WHERE Таб# IN
 (SELECT Таб# FROM С1 WHERE Долж='менеджер') UNION
SELECT Фио
FROM С22
WHERE Таб# IN
 (SELECT Таб# FROM С1 WHERE Долж='менеджер') UNION
SELECT Фио
FROM С23
WHERE Таб# IN
 (SELECT Таб# FROM С1 WHERE Долж='менеджер');
```

В этом случае в запросе необходимо указывать имена используемых фрагментов. Кроме того, дополнительно необходимо воспользоваться операциями соединения (или подзапросами), поскольку атрибуты Долж и Фио находятся в разных вертикальных фрагментах. Основное преимущество прозрачности расположения в том, что база данных может быть подвергнута физической реорганизации, и это не окажет никакого

влияния на программы приложений, осуществляющих к ней доступ. В частности, данные могут быть перенесены с одного узла на другой, и это не должно потребовать внесения каких-либо изменений в использующие их программы или действия пользователей. Такая переносимость желательна, поскольку она позволяет перемещать данные в сети в соответствии с изменяющимися требованиями к эффективности работы системы.

Прозрачность репликации

С прозрачностью расположения очень тесно связан еще один тип прозрачности – прозрачность репликации. Он означает, что пользователю не требуется иметь сведения о существующей репликации фрагментов. Под прозрачностью репликации подразумевается прозрачность расположения реплик. Для пользователей должна быть создана такая среда, чтобы они, по крайней мере, с логической точки зрения, могли считать, что данные не дублируются. Прозрачность репликации (как и прозрачность фрагментации, и прозрачность расположения) является весьма желательной, поскольку она упрощает создание пользовательских программ и выполнение терминальных операций. В частности, прозрачность репликации позволяет создавать и уничтожать дубликаты в любой момент в соответствии с изменяющимися требованиями, не затрагивая при этом никакие из пользовательских приложений или терминальных операций.

Из требований прозрачности репликации следует, что к обязанностям системного оптимизатора также относится определение, какой именно из физических дубликатов будет применен для доступа к данным при выполнении каждого введенного пользователем запроса.

Надо сказать, что могут существовать системы, которые не обеспечивают прозрачности расположения, но поддерживают прозрачность репликации.

Прозрачность локального отображения

Это самый низкий уровень прозрачности распределенности. При наличии в системе прозрачности локального отображения пользователю необходимо указывать как имена используемых фрагментов, так и расположение соответствующих элементов данных. Тот же самый запрос в системе с прозрачностью локального отображения приобретает следующий вид:

```
SELECT Фио
FROM C21 AT SITE 1
WHERE Таб# IN
 (SELECT Таб# FROM C1 AT SITE 1
                  WHERE Долж='менеджер') UNION
SELECT Фио
FROM C22 AT SITE 2
WHERE Таб# IN
 (SELECT Таб# FROM C1 AT SITE 1
                  WHERE Долж='менеджер') UNION
SELECT Фио
FROM C23 AT SITE 3
WHERE Таб# IN
 (SELECT Таб# FROM C1 AT SITE 1
                  WHERE Долж='менеджер');
```

Из соображений наглядности язык SQL дополнен новым ключевым словом *AT SITE*, позволяющим указать, где именно расположен требуемый фрагмент данных. Очевидно, что в данном случае запрос имеет более сложный вид и на его подготовку потребуется больше времени, чем в двух предыдущих случаях. Маловероятно, чтобы система, предоставляющая только такой уровень прозрачности распределенности, была приемлема для конечного пользователя.

Прозрачность именования

Прямым следствием обсуждавшихся выше вариантов прозрачности распределенности является требование наличия **прозрачности именования**. Как и в случае централизованной базы данных, каждый элемент распределенной базы данных должен иметь уникальное имя. Следовательно, РСУБД должна гарантировать, что никакие два узла системы не смогут создать некоторый объект базы данных, имеющий одно и то же имя. Одним из вариантов решения этой проблемы является создание центрального **сервера имен**, который будет нести ответственность за полную уникальность всех существующих в системе имен. Однако подобному подходу свойственны такие недостатки, как:

- утрата определенной части локальной автономии;

- появление проблем с производительностью (поскольку узел с сервером имен превращается в узкое место всей системы);
- снижение доступности – если узел с сервером имен по какой-либо причине станет недоступным, все остальные сайты системы не смогут создавать никаких новых объектов базы данных [2].

Альтернативное решение состоит в использовании префиксов, помещаемых в имена объектов в качестве идентификатора узла, создавшего этот объект [2]. Например, отношение ДЕТАЛИ, созданное на узле $У_1$, могло бы получить имя Yl.ДЕТАЛИ. Аналогичным образом, необходимо иметь возможность идентифицировать каждый фрагмент и каждую его копию. Поэтому второй копии третьего фрагмента отношения ДЕТАЛИ, созданного на узле $У_1$, можно было бы присвоить имя Yl.ДЕТАЛИ.ФЗ.К2. Однако подобный подход приводит к утрате прозрачности распределенности.

Подход, который позволяет преодолеть недостатки, свойственные обоим упомянутым методам, состоит в использовании **алиасов** (вымышленных имен, псевдонимов), создаваемых для каждого из объектов базы данных. В результате объект Yl.ДЕТАЛИ.ФЗ.К2 пользователям узла $У_1$ может быть известен под именем ДЕТАЛИ_ЛОКАЛЬНЫЙ. Задача преобразования алиаса в истинное имя соответствующего объекта базы данных возлагается на РСУБД.

Вопросы

1.30. Какие уровни прозрачности должны поддерживаться РСУБД? Обоснуйте ваш ответ.

1.31. РСУБД должна гарантировать, что на любой паре узлов невозможно будет создать объект базы данных с одним и тем же именем. Одно из решений этой проблемы состоит в организации сервера имен. Какие недостатки свойственны этому подходу? Предложите альтернативный подход, свободный от указанных недостатков.