Лекция 11

Распределенные СУБД

19.1.1. Основные концепции

Чтобы начать обсуждение связанных с распределенными СУБД проблем, прежде всего необходимо уяснить, что же такое распределенная база данных.

Распределенная база данных	Набор логически связанных между собой разделяемых данных (и их описаний), которые физически распределены в некоторой компьютерной сети.

Из этого вытекает следующее определение.

Распределенная СУБД	Программный комплекс, предназначенный для управления распределенными базами данных и позволяющий сделать распределенность информации прозрачной для конечного пользователя.

Система управления распределенными базами данных (СУРБД) состоит из единой логической базы данных, разделенной на некоторое количество **фрагментов**. Каждый фрагмент базы данных сохраняется на одном или нескольких компьютерах, которые соединены между собой линиями связи и каждый из которых работает под управлением отдельной СУБД. Любой из сайтов способен независимо обрабатывать запросы пользователей, требующие доступа к локально сохраняемым данным (что создает определенную степень локальной автономии), а также способен обрабатывать данные, сохраняемые на других компьютерах сети.

Пользователи взаимодействуют с распределенной базой данных через приложения. Приложения могут быть классифицированы как те, которые не требуют доступа к данным на других сайтах (**локальные приложения**), и те, которые требуют подобного доступа (**глобальные приложения**). В распределенной СУБД должно существовать хотя бы одно глобальное приложение, поэтому любая СУРБД должна иметь следующие особенности.

- 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 сайтам.
- Сайты связаны между собой сетевыми соединениями.
- Работа с данными на каждом сайте управляется СУБД.
- СУБД на каждом сайте способна поддерживать автономную работу локальных приложений.
- СУБД каждого сайта поддерживает хотя бы одно глобальное приложение.

Нет необходимости в том, чтобы на каждом из сайтов системы существовала своя собственная локальная база данных, что и показано на примере топологии СУРБД, представленной на рис. 19.1.

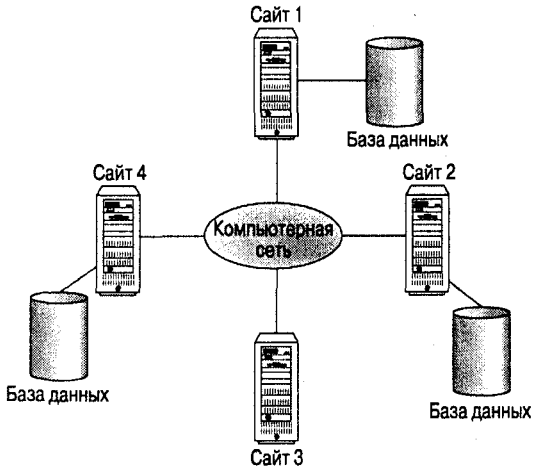


Рис. 19.1. Топология системы управления распределенной базой данных

Пример 19.1. Распределенная обработка данных в компании DreamHome

Используя технологию распределенных баз данных, компания *DreamHome* вместо единственного центрального мейнфрейма может разместить свою базу данных на нескольких независимых компьютерных системах. Подобные компьютерные системы могут быть установлены в каждом из существующих отделений компании — например, в Лондоне, Абердине и Глазго. Сетевые соединения, связывающие компьютерные системы, позволят отделениям компании взаимодействовать между собой, а разворачивание в системе СУРБД обеспечит доступ к данным, размещенным в других отделениях компании. В результате клиент, проживающий в городе Глазго, сможет обратиться в ближайшее отделение компании и ознакомиться с объектами недвижимости, предоставляемыми в аренду в Лондоне, что избавит его от необходимости для получения этих сведений связываться с Лондоном по телефону или посылать почтовое сообщение.

Из определения СУРБД следует, что для конечного пользователя распределенность системы должна быть совершенно **прозрачна** (невидима). Другими словами, от пользователей должен быть полностью скрыт тот факт, что распределенная база данных состоит из нескольких фрагментов, которые могут размещаться на различных компьютерах и для которых, возможно, организована служба репликации данных. Назначение обеспечения прозрачности состоит в том, чтобы распределенная система внешне вела себя точно так, как и централизованная. В некоторых случаях это требование называют **основным принципом** построения распределенных СУБД (Date, 1987). Данный принцип требует предоставления конечному пользователю существенного диапазона функциональных возможностей, но, к сожалению, одновременно ставит перед программным обеспечением СУРБД множество дополнительных задач, с которыми мы подробнее ознакомимся в разделе 19.5.

Распределенная обработка

Очень важно понимать различия, существующие между распределенными СУБД и распределенной обработкой данных.

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

Ключевым 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. 19.2. Сравните этот вариант, содержащий централизованную базу данных на сайте 2, с вариантом, представленным на рис. 19.1, в котором присутствует несколько сайтов, каждый из которых имеет собственную базу данных.

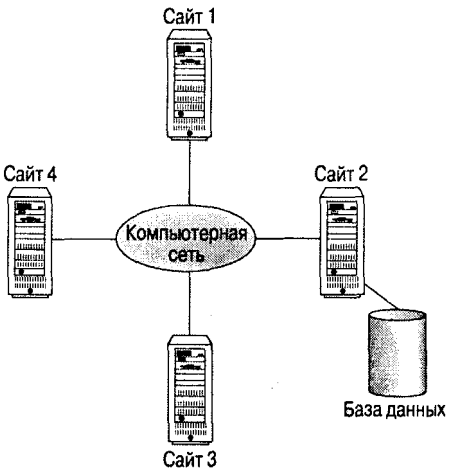


Рис. 19.2. Топология системы с распределенной обработкой

Параллельные СУБД

Кроме того, следует четко понимать различия, существующие между распределенными и параллельными СУБД.

Параллельная СУБД	Система управления базой данных, функционирующая с использованием нескольких процессоров и устройств жестких дисков, что позволяет ей (если это возможно) распараллеливать выполнение некоторых операций с целью повышения общей производительности обработки.

Появление параллельных СУБД было вызвано тем фактом, что системы с одним процессором оказались неспособны удовлетворять растущие требования к масштабируемости, надежности и производительности обработки данных. Эффективной и экономически обоснованной альтернативой однопроцессорным СУБД стали параллельные СУБД, функционирующие одновременно на нескольких процессорах. Применение параллельных СУБД позволяет объединить несколько маломощных машин для получения того же самого уровня производительности, что и в случае одной, но более мощной машины, с дополнительным выигрышем в масштабируемости и надежности системы, по сравнению с однопроцессорными СУБД.

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 и масштабируемости создаваемой системы, что, в свою очередь, определяет пригод-

ность конкретной СУБД к условиям заданной вычислительной среды и требованиям приложений. Три основных типа архитектуры параллельных СУБД представлены на рис. 19.3. К ним относятся:

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

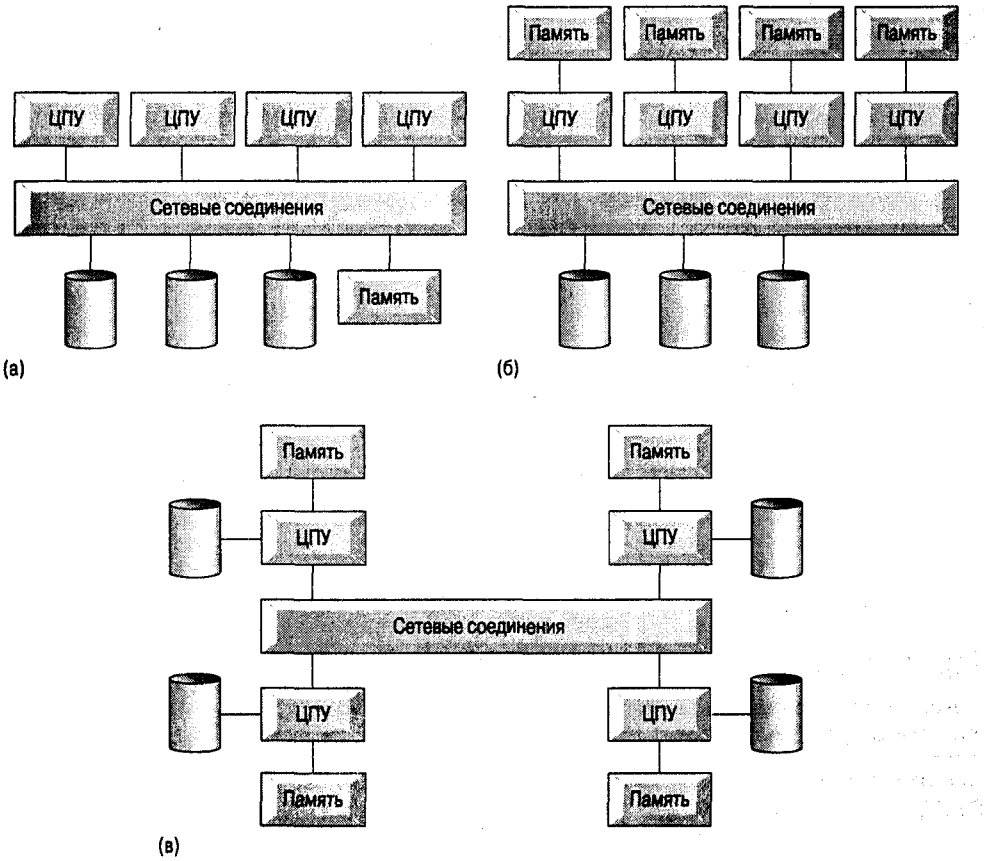


Рис. 19.3. Архитектура систем с параллельной обработкой: а) с разделением памяти; б) с разделением дисков; в) без разделения

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.

Системы с разделением памяти состоят из тесно связанных между собой компонентов, в число которых входит несколько процессоров, разделяющих общую системную память. Иначе называемая симметричной многопроцессорной обработкой (СМП), эта архитектура в настоящее время приобрела большую популярность и применяется для самых разных вычислительных платформ, начиная от персональных рабочих станций, содержащих несколько параллельно работающих микропроцессоров, больших RISC-систем и вплоть до крупнейших мейнфреймов. Данная архитектура обеспечивает быст-

рый доступ к данным для ограниченного числа процессоров, количество которых обычно не превосходит 64. В противном случае сетевые взаимодействия превращаются в узкое место, ограничивающее производительность всей системы.

Системы без разделения (эту архитектуру иначе называют массовой параллельной обработкой — МПП) используют схему, в которой каждый процессор, являющийся частью системы, имеет свою собственную оперативную и дисковую память. База данных распределена между всеми дисковыми устройствами, подключенным к отдельным, связанным с этой базой данных вычислительным подсистемам, в результате чего все данные прозрачно доступны пользователям каждой из этих подсистем. Данная архитектура обеспечивает более высокий уровень масштабируемости, чем системы с СМП, и позволяет легко организовать поддержку работы большого количества процессоров. Однако оптимальной производительности удается достичь только в том случае, если требуемые данные хранятся локально.

Системы с разделением дисков строятся из менее тесно связанных между собой компонентов. Они являются оптимальным вариантом для приложений, которые унаследовали высокую централизацию обработки и должны обеспечивать самые высокие показатели доступности и производительности. Каждый из процессоров имеет непосредственный доступ ко всем совместно используемым дисковым устройствам, но обладает собственной оперативной памятью. Как и в случае архитектуры без разделения, архитектура с разделением дисков исключает узкие места, связанные с совместно используемой памятью. Однако, в отличие от архитектуры без разделения, данная архитектура исключает упомянутые узкие места без внесения дополнительной нагрузки, связанной с физическим распределением данных по отдельным устройствам. Разделяемые дисковые системы в некоторых случаях называют *кластерами*.

Параллельные технологии обычно используются в случае исключительно больших баз данных, размеры которых могут достигать нескольких терабайт (10^{12} байт), или в системах, которые должны поддерживать выполнение тысяч транзакций в секунду. Подобные системы нуждаются в доступе к большому объему данных и должны обеспечивать приемлемое время реакции на запрос. Параллельные СУБД могут использовать различные вспомогательные технологии, позволяющие повысить производительность обработки сложных запросов за счет применения методов распараллеливания операций сканирования, соединения и сортировки, что позволяет нескольким процессорным узлам автоматически распределять между собой текущую нагрузку. Более подробно речь об этих технологиях пойдет в главе 25, "Хранилища данных". В данный момент достаточно отметить, что все крупные разработчики СУБД в настоящее время поставляют параллельные версии созданных ими продуктов.

19.1.2. Преимущества и недостатки, свойственные распределенным СУБД

Системы с распределенными базами данных имеют дополнительные преимущества перед традиционными централизованными системами баз данных. К сожалению, эта технология не лишена и некоторых недостатков. В данном разделе мы обсудим как преимущества, так и недостатки, свойственные системам с распределенными базами данных.

Преимущества

Отражение структуры организации

Крупные организации, как правило имеют множество отделений, которые могут находиться в разных концах страны и даже за ее пределами. Например, компания *DreamHome* имеет многочисленные отделения в различных городах Великобритании. Вполне логично будет предположить, что используемая этой компанией база данных должна быть распределена между ее отдельными офисами. В каждом отделении компании *DreamHome* может поддерживаться база данных, содержащая сведения о

его персонале, сдаваемых в аренду объектах недвижимости, которыми занимаются сотрудники данного отделения, а также о клиентах, которые владеют или желают получить в аренду эти объекты. В подобной базе данных персонал отделения сможет выполнять необходимые ему локальные запросы. Руководству компании может потребоваться выполнять глобальные запросы, предусматривающие получение доступа к данным, сохраняемым во всех существующих отделениях компании.

Разделяемость и локальная автономность

Географическая распределенность организации может быть отражена в распределении ее данных, причем пользователи одного сайта смогут получать доступ к данным, сохраняемым на других сайтах. Данные могут быть помещены на тот сайт, на котором зарегистрированы пользователи, которые их чаще всего используют. В результате заинтересованные пользователи получают локальный контроль над требуемыми им данными и могут устанавливать или регулировать локальные ограничения на их использование. Администратор глобальной базы данных (АБД) отвечает за систему в целом. Как правило, часть этой ответственности делегируется на локальный уровень, благодаря чему АБД локального уровня получает возможность управлять локальной СУБД (см. раздел 4.7).

Повышение доступности данных

В централизованных СУБД отказ центрального компьютера вызывает прекращение функционирования всей СУБД. Однако отказ одного из сайтов СУРБД или линии связи между сайтами сделает недоступным лишь некоторые сайты, тогда как вся система в целом сохранит свою работоспособность. Распределенные СУБД проектируются таким образом, чтобы обеспечивать продолжение функционирования системы несмотря на подобные отказы. Если выходит из строя один из узлов, система сможет перенаправить запросы к отказавшему узлу в адрес другого сайта.

Повышение надежности

Если организована репликация данных, в результате чего данные и их копии будут размещены на более чем одном сайте, отказ отдельного узла или соединительной связи между узлами не приведет к недоступности данных в системе.

Повышение производительности

Если данные размещены на самом нагруженном сайте, который унаследовал от систем-предшественников высокий уровень параллельности обработки, то развертывание распределенной СУБД может способствовать повышению скорости доступа к базе данных (по сравнению с доступом к удаленной централизованной СУБД). Более того, поскольку каждый сайт работает только с частью базы данных, уровень использования центрального процессора и служб ввода/вывода может оказаться ниже, чем в случае централизованной СУБД.

Экономические выгоды

В шестидесятые годы мощность вычислительной установки возрастала пропорционально квадрату стоимости ее оборудования, поэтому система, стоимость которой была втрое выше стоимости данной, превосходила ее по мощности в девять раз. Эта зависимость получила название закона Гроша (Grosch). Однако в настоящее время считается общепринятым положение, согласно которому 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.

Второй потенциальный источник экономии имеет место в том случае, когда базы данных географически удалены друг от друга и приложения требуют осуществления доступа к распределенным данным. В этом случае из-за относительно высокой стоимо-

сти передаваемых по сети данных (по сравнению со стоимостью их локальной обработки) может оказаться экономически выгодным разделить приложение на соответствующие части и выполнять необходимую обработку на каждом из сайтов локально.

Модульность системы

В распределенной среде расширение существующей системы осуществляется намного проще. Добавление в сеть нового сайта не оказывает влияния на функционирование уже существующих. Подобная гибкость позволяет организации легко расширяться. Перегрузки из-за увеличения размера базы данных обычно устраняются путем добавления в сеть новых вычислительных мощностей и устройств дисковой памяти. В централизованных СУБД рост размера базы данных может потребовать замены и оборудования (более мощной системой), и используемого программного обеспечения (более мощной или более гибкой СУБД).

Недостатки

Повышение сложности

Распределенные СУБД, способные скрыть от конечных пользователей распределенную природу используемых ими данных и обеспечить необходимый уровень производительности, надежности и доступности, безусловно являются более сложными программными комплексами, чем централизованные СУБД. Тот факт, что данные могут подвергаться репликации, также добавляет дополнительный уровень сложности в программное обеспечение СУРБД. Если репликация данных не будет поддерживаться на требуемом уровне, система будет иметь более низкий уровень доступности данных, надежности и производительности, чем централизованные системы, а все изложенные выше преимущества превратятся в недостатки.

Увеличение стоимости

Увеличение сложности означает и увеличение затрат на приобретение и сопровождение СУРБД (по сравнению с обычными централизованными СУБД). Разворачивание распределенной СУБД потребует дополнительного оборудования, необходимого для установки сетевых соединений между сайтами. Следует ожидать и роста расходов на оплату каналов связи, вызванных возрастанием сетевого трафика. Кроме того, возрастут затраты на оплату труда персонала, который потребуется для обслуживания локальных СУБД и сетевых соединений.

Проблемы защиты

В централизованных системах доступ к данным легко контролируется. Однако в распределенных системах потребуется организовать контроль доступа не только к данным, реплицируемым на несколько различных сайтов, но и защиту сетевых соединений самих по себе. Раньше сети рассматривались как совершенно незащищенные каналы связи. Хотя это отчасти справедливо и для настоящего времени, тем не менее в отношении защиты сетевых соединений достигнут весьма существенный прогресс.

Усложнение контроля за целостностью данных

Целостность базы данных означает корректность и согласованность сохраняемых в ней данных. Требования обеспечения целостности обычно формулируются в виде некоторых ограничений, выполнение которых будет гарантировать защиту информации в базе данных от разрушения. Реализация ограничений поддержки целостности обычно требует доступа к большому количеству данных, используемых при выполнении проверок, но не требует выполнения операций обновления. В распределенных СУБД повышенная стоимость передачи и обработки данных может препятствовать организации эффективной защиты от нарушений целостности данных. К обсуждению этого вопроса мы вернемся в разделе 20.4.5.

Отсутствие стандартов

Хотя вполне очевидно, что функционирование распределенных СУБД зависит от эффективности используемых каналов связи, только в последнее время стали вырисовываться контуры стандарта на каналы связи и протоколы доступа к данным. Отсутствие стандартов существенно ограничивает потенциальные возможности распределенных СУБД. Кроме того, не существует инструментальных средств и методологий, способных помочь пользователям в преобразовании централизованных систем в распределенные.

Недостаток опыта

В настоящее время в эксплуатации находится уже несколько систем-прототипов и распределенных СУБД специального назначения, что позволило уточнить требования к используемым протоколам и установить круг основных проблем. Однако на текущий момент распределенные системы общего назначения еще не получили широкого распространения. Соответственно, еще не накоплен необходимый опыт промышленной эксплуатации распределенных систем, сравнимый с опытом эксплуатации централизованных систем. Такое положение дел является серьезным сдерживающим фактором для многих потенциальных сторонников данной технологии.

Усложнение процедуры разработки базы данных

Разработка распределенных баз данных, помимо обычных трудностей, связанных с процессом проектирования централизованных баз данных, требует принятия решения о фрагментации данных, распределении фрагментов по отдельным сайтам и организации процедур репликации данных. Все эти проблемы подробнее будут обсуждаться ниже, в разделе 19.4.

Все преимущества и недостатки, свойственные распределенным СУБД, перечислены в табл. 19.1.

Таблица 19.1. Преимущества и недостатки распределенных СУБД

Преимущества	Недостатки
Отображение структуры организации	Повышение сложности
Разделяемость и локальная автономность	Увеличение стоимости
Повышение доступности данных	Проблемы защиты
Повышение надежности	Усложнение контроля за целостностью данных
Повышение производительности	Отсутствие стандартов
Экономические выгоды	Недостаток опыта
Модульность системы	Усложнение процедуры разработки базы данных

19.1.3. Гомогенные и гетерогенные распределенные СУБД

Распределенные СУБД можно классифицировать как гомогенные и гетерогенные. В **гомогенных** системах все сайты используют один и тот же тип СУБД. В **гетерогенных** системах на сайтах могут функционировать различные типы СУБД, использующие разные модели данных, т.е. гетерогенная система может включать сайты с реляционными, сетевыми, иерархическими или объектно-ориентированными СУБД.

Гомогенные системы значительно проще проектировать и сопровождать. Кроме того, подобный подход позволяет поэтапно наращивать размеры системы, последовательно добавляя новые сайты к уже существующей распределенной системе. Дополнительно появляется возможность повышать производительность системы за счет организации на различных сайтах параллельной обработки информации.

Гетерогенные системы обычно возникают в тех случаях, когда независимые сайты, уже эксплуатирующие свои собственные системы с базами данных, интегрируются во вновь создаваемую распределенную систему. В гетерогенных системах для организации

взаимодействия между различными типами СУБД потребуется организовать трансляцию передаваемых сообщений. Для обеспечения прозрачности в отношении типа используемой СУБД пользователи каждого из сайтов должны иметь возможность вводить интересующие их запросы на языке той СУБД, которая используется на данном сайте. Система должна взять на себя локализацию требуемых данных и выполнение трансляции передаваемых сообщений. В общем случае данные могут быть затребованы с другого сайта, который характеризуется такими особенностями, как:

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

Если используется иной тип оборудования, однако на сайте установлен тот же самый тип СУБД, методы выполнения трансляции вполне очевидны и включают замену кодов и изменение длины слова. Если типы используемых на сайтах СУБД различны, процедура трансляции усложняется тем, что необходимо отображать 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, потребуется транслировать текст запросов с одного используемого языка на другой (например, запросы с SQL-оператором SELECT потребуется преобразовать в запросы с операторами FIND и GET языка манипулирования данными сетевой СУБД). Если отличаются и тип используемого оборудования, и тип программного обеспечения, потребуется выполнять оба вида трансляции. Все изложенное выше чрезвычайно усложняет обработку данных в гетерогенных СУБД.

Дополнительные сложности возникают при попытках выработки единой концептуальной схемы, создаваемой путем интеграции независимых локальных концептуальных схем. Как уже указывалось при обсуждении этапа 3.1 предложенной в главе 8 методологии логического проектирования баз данных, при наличии семантической неоднородности интеграция локальных моделей данных превращается в чрезвычайно трудную задачу. Например, атрибуты, имеющие в различных схемах одно и то же имя, на деле могут представлять совершенно отличные понятия. Аналогично, атрибуты с разными именами фактически могут представлять одну и ту же характеристику. Подробное обсуждение методов выявления и устранения семантической неоднородности выходит за рамки данной книги. Заинтересованному читателю мы рекомендуем обратиться к работе Гарсиа-Солако и др. (Garcia-Solaco et al., 1996).

Типичное решение, применяемое в некоторых реляционных системах, состоит в том, что отдельные части гетерогенных распределенных систем должны использовать шлюзы, предназначенные для преобразования языка и модели данных каждого из используемых типов СУБД в язык и модель данных реляционной системы. Однако подходу с использованием шлюзов свойственны некоторые серьезные ограничения. Во-первых, шлюзы не позволяют организовать систему управления транзакциями даже для отдельных пар систем. Другими словами, шлюз между двумя системами представляет собой не более чем транслятор запросов. Например, шлюзы не позволяют системе координировать управление параллельностью и процедурами восстановления транзакций, включающих обновление данных в обеих базах. Во-вторых, использование шлюзов призвано лишь решить задачу трансляции запросов с языка одной СУБД на язык другой СУБД. Поэтому они не позволяют справиться с проблемами, вызванными неоднородностью структур и представлением данных в различных схемах.

Открытый доступ и взаимодействие баз данных

Комитет Open Group организовал рабочую группу (Specification Working Group — SWG), призванную подготовить ответ на поступающие запросы по поводу открытого доступа и взаимодействия баз данных (Gualteri, 1996). Цель работы этой группы состоит в подготовке спецификаций (или в получении подтверждений того, что требуемые спецификации существуют или разрабатываются), регламентирующих инфраструктуру среды базы данных, включающую следующие элементы.

- Унифицированный и достаточно мощный интерфейс языка SQL (SQL API), позволяющий создавать клиентские приложения таким образом, 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.

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

Мультибазовые системы

Прежде чем завершить данный раздел, целесообразно кратко ознакомиться с одной из разновидностей распределенных СУБД, называемой мультибазовой системой.

Мультибазовая система	*Распределенная система управления базами данных, в которой* управление каждым из сайтов осуществляется совершенно автономно.

В последние годы заметно возрос интерес к мультибазовым СУБД, в которых предпринимается попытка интеграции таких распределенных систем баз данных, в которых весь контроль над отдельными локальными системами целиком и полностью осуществляется их операторами. Одним из следствий полной автономии сайтов является отсутствие необходимости внесения каких-либо изменений 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 сайтов получать доступ и совместно использовать данные без необходимости физической интеграции существующих баз данных. Они обеспечивают пользователям возможность управлять базами данных их собственных сайтов без какого-либо централизованного контроля, который обязательно присутствует в обычных типах СУРБД. 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. Существуют **нефедеральные** (не имеющие локальных пользователей) и **федеральные** мультибазовые системы. Федеральная система представляет собой некоторый гибрид распределенной и централизованной систем, поскольку она выглядят как распределенная система для удаленных пользователей и как централизованная система — для локальных. Информацию о классификации распределенных систем заинтересованный читатель сможет найти в работах Шета и Ларсона (Sheth and Larson, 1990), а также Букреса и Элмагармида (Bukhres and Elmagarmid, 1996).

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. Мультибазовая СУБД поддерживает глобальную схему, на основании которой пользователи могут 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. Затем полученные после выполнения локальных запросов результаты сливаются в единый глобальный результат, предоставляемый пользователю. Кроме того, мульти-

базовая 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 они контролируют работу локальных СУБД.

Так, мультибазовая система UniSQL/M фирмы UniSQL Inc. позволяет разрабатывать приложения с помощью единого глобального представления и единственного языка доступа к базе данных для работы со многими гетерогенными реляционными и объектно-ориентированными СУБД (Connolly et al., 1994). Подробнее речь о мультибазовых СУБД пойдет в разделе 19.3.3.

19.2. Принципы организации и работы компьютерных сетей

Компьютерная сеть	Множество автономных компьютеров, соединенных между собой и способных обмениваться информацией.

Компьютерные сети представляют собой сложную и интенсивно развивающуюся область компьютерной индустрии, однако определенные знания в этой области совершенно необходимы для понимания принципов построения распределенных СУБД. За последних несколько десятилетий был пройден путь от использования полностью автономных отдельных компьютеров до современного состояния, когда сети компьютеров получили повсеместное распространение. Размеры компьютерных сетей варьируются от нескольких соединенных между собой персональных компьютеров до сетей глобального масштаба, насчитывающих тысячи машин и сотни тысяч пользователей. В нашем конкретном случае распределенные СУБД устанавливаются поверх компьютерной сети таким образом, что работа последней оказывается полностью скрыта от конечного пользователя.

Сетевые соединения можно классифицировать несколькими различными способами. Можно классифицировать сети, взяв за основу расстояние между компьютерами: если оно невелико, сеть называется **локальной**, в противном случае — **глобальной**. Локальные сети (local area network — LAN) образуются при соединении компьютеров, принадлежащих одному и тому же сайту. Глобальные сети (wide area network — WAN) используются для соединения компьютеров различных локальных сетей, находящихся на удаленном расстоянии друг от друга. Благодаря большей географической удаленности, линии соединения глобальных сетей обладают относительно невысокой пропускной способностью и меньшей надежностью по сравнению с локальными сетями. Скорость передачи данных в глобальных сетях обычно находится в пределах от 2 до 2000 Кбит/с. Скорость передачи данных в локальных сетях намного больше и составляет от 10 до 100 Мбит/с, а надежность установленных соединений существенно выше. Очевидно, что распределенные системы, построенные на основе локальных сетей, будут обеспечивать меньшее время реакции системы, чем системы, использующие глобальные сети.

Если классифицировать сети на основе метода выбора пути (или **маршрутизации**), то их можно разделить на **широковещательные** и **одноадресные**. В случае одноадресной сети при необходимости разослать сообщения всем узлам сети потребуется отправить несколько отдельных сообщений. В случае широковещательной сети все узлы получают все сообщения, однако в каждом из сообщений присутствует префикс, идентифицирующий узел получатель, а все остальные узлы сети просто проигнорируют поступившее сообщение. Глобальные сети обычно строятся по одноадресному принципу, тогда как в локальных сетях, как правило, используется широковещательный принцип. Сводка типичных характеристик глобальных и локальных сетей представлена в табл. 19.2.

Таблица 19.2. Сравнение характеристик глобальных и локальных сетей

Глобальные сети	Локальные сети
Удаленность узлов до тысяч километров	Удаленность узлов до нескольких километров
Связывают автономные компьютеры	Связывают компьютеры, совместно использующие распределенные приложения
Сеть управляется независимой организацией (используются телефонные или спутниковые каналы связи)	Сеть управляется ее пользователями (используются собственные кабельные соединения)
Скорость передачи данных до 2 Мбит/с (линии T1) или 45 Мбит/с (линии T3)	Скорость передачи данных до 100 Мбит/с
Сложные протоколы	Более простые протоколы
Используется одноадресная маршрутизация	Используется широковещательная маршрутизация
Используется нерегулярная топология	Используется топология шины или звезды
Вероятность ошибки порядка $1:10^5$	Вероятность ошибки порядка $1:10^9$

Международная организация стандартов (ISO) установила набор правил (или протоколов), регламентирующих способы взаимодействия систем (ISO, 1981). Выбранный подход состоит в разделении сетевого аппаратного и программного обеспечения на несколько уровней, каждый из которых предоставляет определенные услуги расположенным выше уровням, одновременно скрывая от них все подробности реализации нижних уровней. Протокол, получивший название "модель OSI" (Open System Interconnection), предусматривает использование семи уровней, логически не зависящих от изготовителя оборудования или программ. Отдельные уровни отвечают за передачу последовательностей битов информации по сети, за установку соединений и контроль наличия ошибок, маршрутизацию и устранение заторов в сети, организацию сеансов связи между отдельными машинами и устранение различий в формате и способе представления данных в компьютерах различных платформ. Полное описание особенностей этого протокола не является необходимым для понимания материала оставшейся части этой главы и следующей главы, посвященной управлению распределенными транзакциями, поэтому тем, кто желает подробнее узнать об этом, мы рекомендуем обратиться к работам Халсалла (Halsall, 1995) и Таненбаума (Tanenbaum, 1996).

Международный консультативный комитет по телеграфу и телефонии (CCITT) разработал стандарт, получивший название X.25 и охватывающий три нижних уровня описанной выше модели. Большинство распределенных СУБД было разработано с целью использования поверх протокола X.25. Однако позже были выпущены новые стандарты, охватывающие более высокие уровни модели и способные предоставить СУРБД полезные дополнительные функциональные возможности. К ним можно отнести протоколы RDA (Remote Database Access — ISO, 9579) и DTP (Distribution Transaction Processing — ISO, 10026). Со стандартом X/Open DTP мы познакомимся в разделе 20.5.

Время передачи

Время, необходимое для доставки сообщения, зависит от размеров посылаемого сообщения и типа используемого сетевого соединения. Оно может быть определено по следующей формуле:

время_передачи = C_0 + (*количество_бит_сообщения*/*скорость_передачи*)

Здесь константа C_0 представляет затраты времени на инициацию сообщения, называемые **задержкой доступа**. Например, в случае задержки доступа, равной 1 с, и скорости передачи — 10 000 бит/с время, необходимое для отправки 100 000 записей размером 100 байт, составит:

```
время_передачи = 1 + (100 000 * 100/10 000) = 1001 сек.
```

Если необходимо отправлять записи по одной, то время передачи составит:

```
время_передачи = 100 000 * [1 + (100/10 000)]
               = 100 000 * [1.01] = 101 000 сек.
```

Очевидно, что время передачи существенно выше при передаче каждой из 100 000 записей в отдельности, а не единым пакетом, что вызвано наличием задержки доступа. Соответственно, задачей СУРБД является минимизация как времени передачи данных по сети, так и сокращение количества передач данных. Мы вернемся к обсуждению этого вопроса позже, при рассмотрении оптимизации распределенных запросов в разделе 19.5.3.

19.3. Функции и архитектура распределенных СУБД

В главе 2 мы познакомились с функциями, архитектурой и компонентами централизованных СУБД. В этом разделе мы обсудим, какое влияние распределенность данных оказывает на требуемый набор функциональных возможностей и архитектуру СУРБД.

19.3.1. Функции распределенных СУБД

Следует ожидать, что типичная СУРБД должна обеспечивать, по крайней мере, тот же набор функциональных возможностей, который был определен нами для централизованных СУБД в главе 2. Кроме того, СУРБД должна предоставлять следующий набор функциональных возможностей.

- Расширенные службы установки соединений должны обеспечивать доступ к удаленным сайтам и позволять передавать запросы и данные между сайтами, входящими в сеть.
- 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 сайтов и отказов линий связи.

Все эти проблемы будут подробно обсуждаться позже, в последующих разделах этой и следующей глав.

19.3.2. Рекомендуемая архитектура распределенных СУБД

Трехуровневая архитектура ANSI-SPARC для СУБД, обсуждавшаяся в разделе 2.1, представляет собой типовое решение для централизованных СУБД. Однако распределенные СУБД имеют множество отличий, которые весьма сложно отразить в некотором эквивалентном архитектурном решении, приемлемом для большинства случаев. Однако было бы полезно найти некоторое рекомендуемое решение, учитывающие особенности работы с распределенными данными. Один из примеров рекомендуемой архитектуры СУРБД представлен на рис. 19.4. Он включает следующие элементы:

- 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

Глобальная концептуальная схема представляет собой логическое описание всей базы данных, представляющее ее так, как будто она не является распределенной. Этот уровень СУРБД 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.

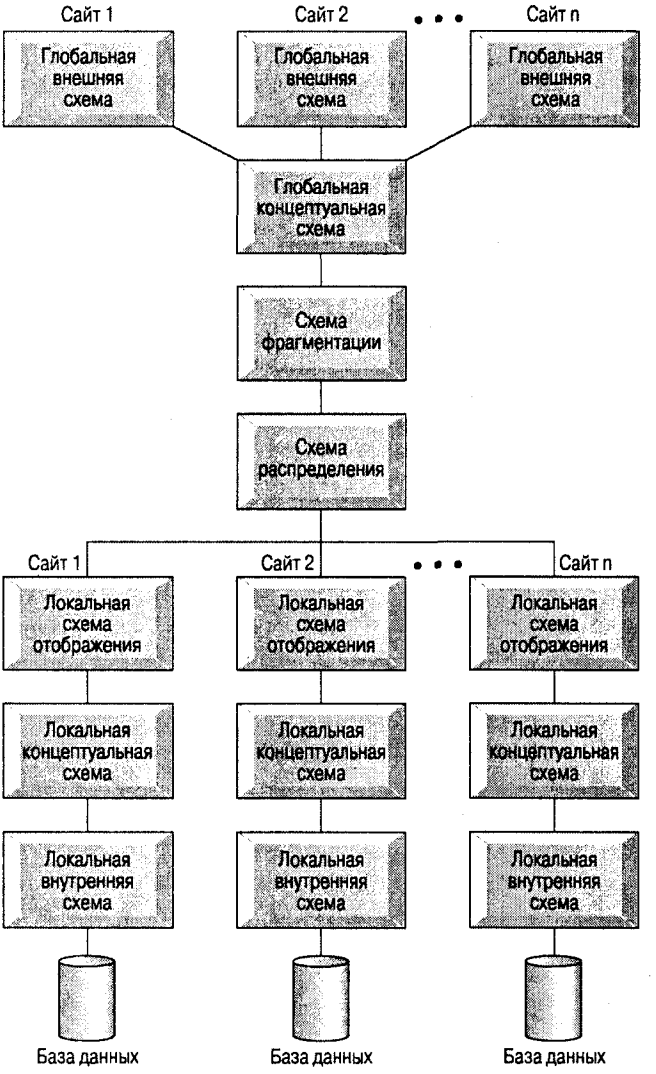


Рис. 19.4. Архитектура, рекомендуемая для СУРБД

Схемы фрагментации и распределения

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.

Локальные схемы

Каждая локальная СУБД имеет свой собственный набор схем. Локальная концептуальная и локальная внутренняя схемы полностью соответствуют эквивалентным 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 СУРБД.

19.3.3. Рекомендуемая архитектура мультибазовых СУБД

В разделе 19.1.3 было приведено краткое обсуждение федеральных мультибазовых систем. Подобные системы отличаются от распределенных предоставляемым уровнем локальной автономности. Это отличие также должно быть учтено в архитектуре, рекомендуемой для систем данного типа. На рис. 19.5 показана архитектура, рекомендуемая для **тесно связанных** федеральных мультибазовых систем, использующих глобальную концептуальную схему. В распределенных СУБД глобальная концептуальная схема является объединением всех локальных концептуальных схем. В федеральных мультибазовых СУБД глобальная концептуальная схема является подмножеством локальных концептуальных схем, включающим только те данные, которые каждая из локальных систем разрешает использовать совместно. Глобальная концептуальная схема тесно связанных систем содержит интегрированное представление либо элементов локальных концептуальных схем, либо локальных внешних схем.

Существует мнение, что федеральные мультибазовые СУБД не должны использовать глобальные концептуальные схемы (Litwin, 1988). В этом случае система будет называться **слабо связанной**. В подобных системах внешние схемы состоят из одной или нескольких локальных концептуальных схем. Дополнительную информацию о мультибазовых СУБД заинтересованный читатель сможет найти в публикациях Литвина (Litwin, 1988), а также Шета и Ларсона (Sheth and Larson, 1990).

19.3.4. Компонентная архитектура распределенных СУБД

Независимо от обсуждавшейся выше рекомендованной общей архитектуры СУРБД следует рассмотреть компонентную архитектуру СУРБД, которая должна включать четыре следующих важнейших компонента:

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

Общий вид компонентной архитектуры распределенной СУБД со схемой, показанной на рис. 19.1, представлен на рис. 19.6. Для упрощения сайт 2 на этой диаграмме опущен, поскольку его структура не отличается от структуры сайта 1.

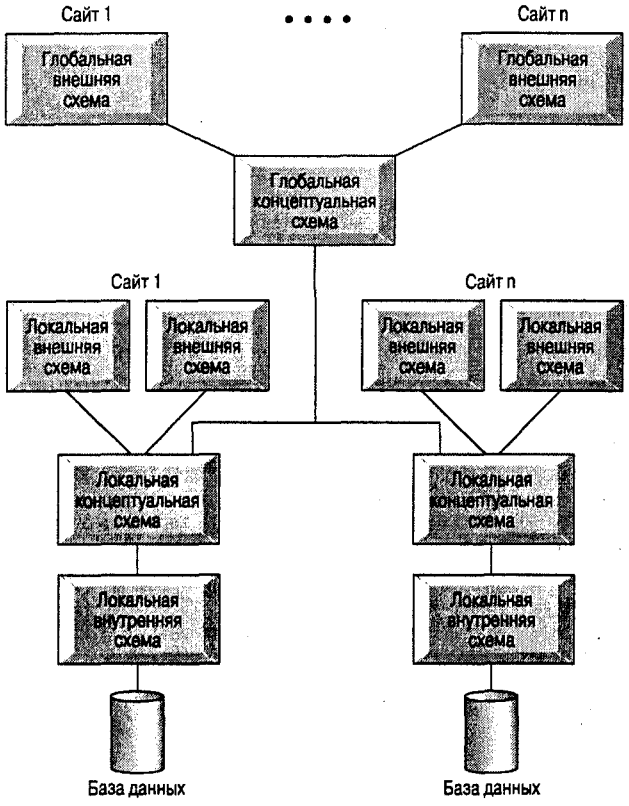


Рис. 19.5. Архитектура, рекомендуемая для тесно связанных федеральных мультибазовых СУБД

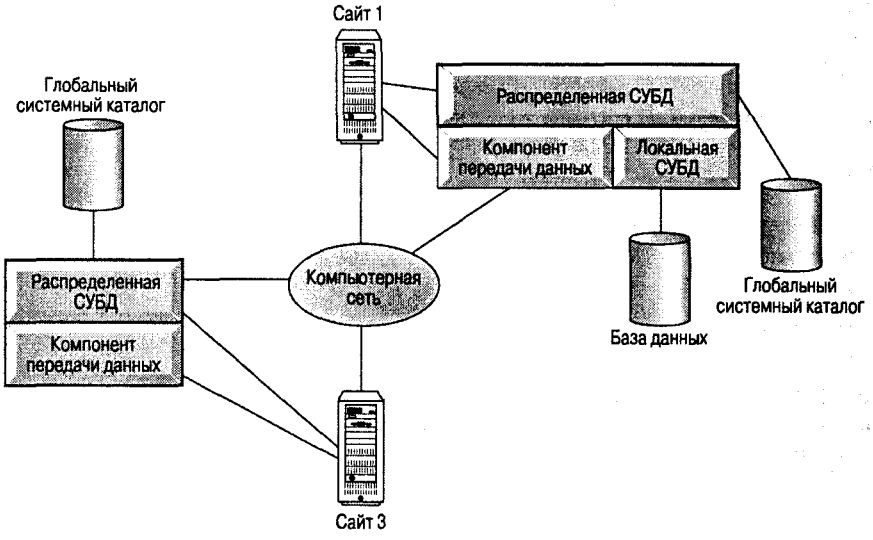


Рис. 19.6. Компонентная архитектура распределенной СУБД

Локальная СУБД

Компонент локальной СУБД представляет собой стандартную СУБД, предназначенную для управления локальными данными на каждом из сайтов, входящих в состав распределенной базы данных. Локальная СУБД имеет свой собственный системный каталог, в котором содержится информация о данных, сохраняемых на этом сайте. В гомогенных системах на каждом из сайтов в качестве локальной СУБД используется один и тот же программный продукт. В гетерогенных системах существуют, по крайней мере, два сайта, использующих различные типы СУБД и/или различные типы вычислительных платформ.

Компонент передачи данных

Компонент передачи данных представляет собой программное обеспечение, позволяющее всем сайтам взаимодействовать между собой. Он содержит сведения о существующих сайтах и линиях связи между ними.

Глобальный системный каталог

Глобальный системный каталог имеет то же самое функциональное назначение, что и системный каталог в централизованных базах данных. 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 подвергаться фрагментации и распределению, быть полностью реплицируемым или централизованным, как и любое другое отношение, о чем речь пойдет ниже. Полностью реплицируемый глобальный системный каталог снижает автономность отдельных сайтов, поскольку любые изменения в нем должны передаваться на все существующие сайты. Использование централизованного глобального каталога также снижает автономность сайтов и повышает их зависимость от отказов на центральном сайте. Подход, выбранный в системе R*, позволяет преодолеть все указанные недостатки. В системе R* на каждом сайте существует локальный каталог, содержащий метаданные, описывающие данные, сохраняемые на этом сайте. Что касается отношений, созданных на некотором сайте (*сайте создания*), то ответственность за фиксацию описания каждого его фрагмента, каждой реплики каждого фрагмента, а также хранение сведений о расположении этих фрагментов возлагается на локальный каталог данного сайта. В случае, если фрагмент или реплика перемещается в другое место, сведения в локальном каталоге сайта создания соответствующего отношения необходимым образом обновляются. Следовательно, для определения расположения фрагмента или реплики отношения необходимо получить доступ к каталогу его сайта создания. Сведения о сайте создания каждого глобального отношения должны фиксироваться в каждом локальном экземпляре глобального системного каталога.

Распределенная СУБД

Компонент распределенной СУБД является управляющим по отношению ко всей системе элементом. В предыдущем разделе мы кратко познакомились с основными функциональными возможностями, которыми должен обладать этот компонент. Обсуждение этих функциональных возможностей мы продолжим в оставшейся части этой главы и в следующей главе.

19.4. Разработка распределенных реляционных баз данных

В главах 7 и 8 вашему вниманию была предложена методология концептуального и логического проектирования централизованных реляционных баз данных. В данном разделе мы рассмотрим дополнительные факторы, которые должны приниматься во внимание при разработке распределенных реляционных баз данных. В частности, мы обсудим следующие аспекты проектирования распределенных систем.

- **Фрагментация.** Любое отношение может быть разделено на некоторое количество частей, называемых фрагментами, которые затем распределяются по различным сайтам. Существуют два основных типа фрагментов: **горизонтальные** и **вертикальные**. Горизонтальные фрагменты представляют собой подмножества кортежей, а вертикальные — подмножества атрибутов.
- **Распределение.** Каждый фрагмент сохраняется на сайте, выбранном с учетом "оптимальной" схемы их размещения.
- **Репликация.** СУРБД может поддерживать актуальную копию некоторого фрагмента на нескольких различных сайтах.

Определение и размещение фрагментов должно проводиться с учетом особенностей использования базы данных. В частности, это подразумевает выполнение анализа приложений. Как правило, провести анализ всех приложений не представляется возможным, поэтому следует сосредоточить усилия на самых важных из них. Как уже было отмечено в разделе 9.2, эмпирически отмечено, что 20% выполняемых пользователями запросов создают 80% общей нагрузки на базу данных. Это же правило "80/20" вполне может использоваться при проведении анализа приложений (Weiderhold, 1983).

Проектирование должно выполняться на основе как количественных, так и качественный показателей. Количественная информация используется в качестве основы для распределения, тогда как качественная служит базой при создании схемы фрагментации. Количественная информация включает такие показатели:

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

Качественная информация может включать перечень выполняемых в приложении транзакций, используемые отношения, атрибуты и кортежи, к которым осуществляется доступ, тип доступа (чтение или запись), предикаты, используемые в операциях чтения.

Определение и размещение фрагментов по сайтам выполняется для достижения следующих стратегических целей.

- *Локальность ссылок.* Везде, где только это возможно, данные должны храниться как можно ближе к местам их использования. Если фрагмент используется на нескольких сайтах, может оказаться целесообразным разместить на этих сайтах его копии.
- *Повышение надежности и доступности.* Надежность и доступность данных повышаются за счет использования механизма репликации. В случае отказа одного из сайтов всегда будет существовать копия фрагмента, сохраняемая на другом сайте.
- *Приемлемый уровень производительности.* Неверное распределение данных будет иметь следствием возникновение в системе узких мест. В этом случае некоторый сайт оказывается просто завален запросами со стороны других сайтов, что может вызвать существенное снижение производительности всей системы. В то же время неправильное распределение может иметь следствием неэффективное использование ресурсов системы.
- *Баланс между емкостью и стоимостью внешней памяти.* Обязательно следует учитывать доступность и стоимость устройств хранения данных, имеющихся на каждом из сайтов системы. Везде, где только это возможно, рекомендуется использовать более дешевые устройства массовой памяти. Это требование должно быть сбалансировано с требованием поддержки *локальности ссылок*.
- *Минимизация расходов на передачу данных.* Следует тщательно учитывать стоимость выполнения в системе удаленных запросов. Затраты на выборку будут минимальны при обеспечении максимальной *локальности ссылок*,

т.е. тогда, когда каждый сайт будет иметь собственную копию данных. Однако при обновлении реплицируемых данных внесенные изменения потребуется распространить на все сайты, имеющие копию обновленного отношения, что вызовет увеличение затрат на передачу данных.

19.4.1. Распределение данных

Существуют четыре альтернативные стратегии размещения данных в системе: централизованное, раздельное (фрагментированное), размещение с полной репликацией и с выборочной репликацией. Ниже мы узнаем, какие результаты дает использование каждой из этих стратегий, и рассмотрим их в свете достижения целей, определенных выше.

Централизованное размещение

Данная стратегия предусматривает создание на одном из сайтов единственной базы данных под управлением СУБД, доступ к которой будут иметь все пользователи сети (эта стратегия под названием "распределенная обработка" уже рассматривалась нами выше.) В этом случае локальность ссылок минимальна для всех сайтов, за исключением центрального, поскольку 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 отказ на центральном сайте вызовет паралич работы всей системы.

Раздельное (фрагментированное) размещение

В этом случае база данных разбивается на непересекающиеся фрагменты, каждый из которых размещается на одном из сайтов системы. Если элемент данных будет размещен на том сайте, на котором он чаще всего используется, полученный уровень локальности ссылок будет высок. При отсутствии репликации стоимость хранения данных будет минимальна, но при этом будет невысок также уровень надежности и доступности данных в системе. Однако он будет выше, чем в предыдущем варианте, поскольку отказ на любом из сайтов вызовет утрату доступа только к той части данных, которая на нем хранилась. При правильно выбранном способе распределения данных уровень производительности в системе будет относительно высоким, а уровень затрат на передачу данных — низким.

Размещение с полной репликацией

Эта стратегия предусматривает размещение полной копии всей базы данных на каждом из сайтов системы. Следовательно, локальность ссылок, надежность и доступность данных, а также уровень производительности системы будут максимальны. Однако стоимость устройств хранения данных и уровень затрат на передачу данных в этом случае также будут самыми высокими. Для преодоления части этих проблем в некоторых случаях используется технология моментальных снимков. **Моментальный снимок** представляет собой копию базы данных в определенный момент времени. Эти копии обновляются через некоторый установленный интервал времени — например, один раз в час или в неделю, — а потому они не всегда будут актуальными в текущий момент. Иногда в распределенных системах моментальные снимки используются для реализации представлений, что позволяет улучшить время выполнения в базе данных операций с представлениями. Подробнее о моментальных снимках речь пойдет в разделе 20.6.

Размещение с выборочной репликацией

Данная стратегия представляет собой комбинацию методов фрагментации, репликации и централизации. Одни массивы данных разделяются на фрагменты, что позволяет добиться для них высокой локальности ссылок, тогда как другие, используемые на многих сайтах, но не подверженные частым обновлениям, подвергаются

репликации. Все остальные данные хранятся централизованно. Целью применения данной стратегии является объединение всех преимуществ, существующих в остальных моделях, с одновременным исключением свойственных им недостатков. Благодаря своей гибкости именно эта стратегия используется чаще всего. Сводные характеристики всех рассмотренных выше стратегий приведены в табл. 19.3. Дополнительные сведения о распределении данных заинтересованный читатель найдет в работах Оцзу и Валдуриза (Ozsu and Valduriez, 1997), а также Теорея (Teorey, 1994).

Таблица 19.3. Сравнительные характеристики различных стратегий распределения данных

	Локальность ссылок	Надежность и доступность	Производительность	Стоимость устройств хранения	Затраты на передачу данных
Централизованное	Самая низкая	Самая низкая	Неудовлетворительная	Самая низкая	Самая высокая
Фрагментированное	Высокая[1]	Низкая для отдельных элементов; высокая для системы в целом	Удовлетворительная[1]	Самая низкая	Низкая[1]
Полная репликация	Самая высокая	Самая высокая	Хорошая для операций чтения	Самая высокая	Высокая для операций обновления, низкая для операций чтения
Выборочная репликация	Высокая[1]	Низкая для отдельных элементов, высокая для системы	Удовлетворительная[1]	Средняя	Низкая[1]

[1] При условии качественного проектирования.

19.4.2. Фрагментация

Назначение фрагментации

Прежде чем приступить к подробному обсуждению различных аспектов фрагментации, целесообразно ознакомиться с причинами, вызывающими необходимость фрагментации отношений.

- *Условия использования.* Чаще всего приложения работают с некоторыми представлениями, а не с полными базовыми отношениями. Следовательно, с точки зрения распределения данных, целесообразнее организовать работу приложений с определенными фрагментами отношений, выступающими как распределяемые элементы.
- *Эффективность.* Данные хранятся в тех местах, в которых они чаще всего используются. Кроме того, исключается хранение данных, которые не используются локальными приложениями.
- *Параллельность.* Поскольку фрагменты являются распределяемыми элементами, транзакции могут быть разделены на несколько подзапросов, обращающихся к различным фрагментам. Такой подход дает возможность повысить уровень параллельности обработки в системе, т.е. позволяет транзакциям, которые допускают это, безопасно выполняться в параллельном режиме.
- *Защищенность.* Данные, не используемые локальными приложениями, не хранятся на сайтах, а значит, неавторизированные пользователи не смогут получить к ним доступ.

Механизму фрагментации свойственны два основных недостатка, которые уже упоминались выше.

- *Производительность*. Производительность приложений, требующих доступа к данным из нескольких фрагментов, расположенных на различных сайтах, может оказаться недостаточной.
- *Целостность данных*. Поддержка целостности данных может существенно осложняться, поскольку функционально зависимые данные могут оказаться фрагментированными и размещаться на различных сайтах.

Корректность фрагментации

Фрагментация данных не должна выполняться непродуманно, наугад. Существуют три правила, которых следует обязательно придерживаться при проведении фрагментации.

1. *Полнота*. Если экземпляр отношения R разбивается на фрагменты, например R_1, R_2, ..., R_n, то каждый элемент данных, присутствующий в отношении R, должен присутствовать, по крайней мере, в одном из созданных фрагментов. Выполнение этого правила гарантирует, что какие-либо данные не будут утрачены в результате выполнения фрагментации.
2. *Восстановимость*. Должна существовать операция реляционной алгебры, позволяющая восстановить отношение R из его фрагментов. Это правило гарантирует сохранение функциональных зависимостей.
3. *Непересекаемость*. Если элемент данных d_i присутствует во фрагменте R_i, то он не должен одновременно присутствовать в каком-либо ином фрагменте. Исключением из этого правила является операция вертикальной фрагментации, поскольку в этом случае в каждом фрагменте должны присутствовать атрибуты первичного ключа, необходимые для восстановления исходного отношения. Данное правило гарантирует минимальную избыточность данных во фрагментах.

В случае горизонтальной фрагментации элементом данных является кортеж, а в случае вертикальной фрагментации — атрибут.

Типы фрагментации

Существуют два основных типа фрагментации: **горизонтальная** и **вертикальная**. Горизонтальные фрагменты представляют собой подмножества кортежей отношения, а вертикальные — подмножества атрибутов отношения, как показано на рис. 19.7.

Кроме того, существуют еще два типа фрагментации: **смешанная** (рис. 19.8) и **производная** (представляющая собой вариант горизонтальной фрагментации). Ниже мы продемонстрируем различные типы фрагментации на примере экземпляра базы данных приложения *DreamHome*, представленного в табл. 3.3–3.8.

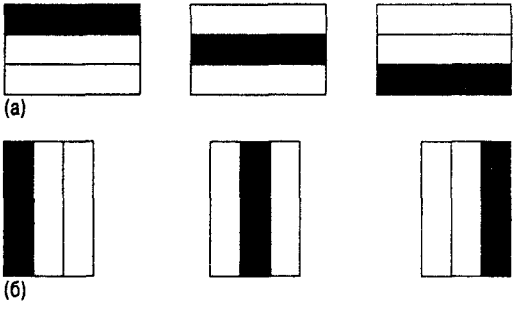


Рис. 19.7. Различные типы фрагментации: а) горизонтальная; б) вертикальная

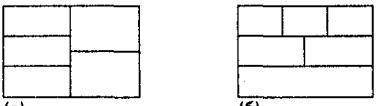


Рис. 19.8. Смешанная фрагментация: а) горизонтально разделенные вертикальные фрагменты; б) вертикально разделенные горизонтальные фрагменты

Горизонтальная фрагментация

Горизонтальный фрагмент	Выделенный по горизонтали фрагмент отношения, состоящий из некоторого подмножества кортежей этого отношения.

Горизонтальный фрагмент создается посредством определения предиката, с помощью которого выполняется отбор кортежей из исходного отношения. Данный тип фрагмента определяется с помощью операции *выборки* реляционной алгебры (см. раздел 3.4.1). Операция *выборки* позволяет выделить группу кортежей, обладающих некоторым общим для них свойством, — например, все кортежи, используемые одним из приложений, или все кортежи, применяемые на одном из сайтов. Если задано отношение R, то его горизонтальный фрагмент может быть определен с помощью следующей формулы:

$\sigma_p(R)$

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

Пример 19.2. Горизонтальная фрагментация

Предположим, что существуют только два типа объектов недвижимости: квартира ('Flat') и дом ('House'). В этом случае горизонтальная фрагментация отношения Property_for_Rent по атрибуту Type может быть выполнена следующим образом:

P_1: $\sigma_{type='House'}$(Property_for_Rent)

P_2: $\sigma_{type='Flat'}$(Property_for_Rent)

В результате будут созданы два фрагмента. Первый, содержимое которого представлено в табл. 19.4, будет состоять из кортежей, в которых значение атрибута Type будет равно 'House'. Второй фрагмент (табл. 19.5) будет состоять из кортежей, в которых значение атрибута Type равно 'Flat'. Подобный вариант фрагментации может оказаться полезным, если существуют независимые приложения, обрабатывающие данные только о квартирах или домах. Предложенная схема фрагментации отвечает всем правилам корректности.

- *Полнота*. Каждый кортеж исходного отношения присутствует либо во фрагменте P_1, либо во фрагменте P_2.
- *Восстановимость*. Отношение Property_for_Rent может быть восстановлено из созданных фрагментов с помощью следующей операции объединения:

  $P_1 \cup P_2$ = Property_for_Rent

- *Непересекаемость*. Полученные фрагменты не пересекаются, поскольку не существует значения атрибута Type, которое одновременно было бы равно значениям 'House' и 'Flat'.

Таблица 19.4. Горизонтальный фрагмент P_1 отношения Property_for_Rent

Pno	Sreet	Area	City	Pcode	Type	Rooms	Rent	Ono	Sno	Bno
PA14	16 Holhead	Dee	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B7
PG21	18 Dale Rd	Hyndland	Glasgow	G12	House	5	600	CO87	SG37	B3

Таблица 19.5. Горизонтальный фрагмент P_2 отношения Property_for_Rent

Pno	Sreet	Area	City	Pcode	Type	Rooms	Rent	Ono	Sno	Bno
PL94	6 Argyll St	Kilburn	London	NW2	Flat	4	400	CO87	SL41	B5
PG4	6 Lawrence St	Partick	Glasgow	G11 9QX	Flat	3	350	CO40	SG14	B3
PG36	2 Manor Rd		Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B3
PG16	5 Novar Dr	Hyndland	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B3

В одних случаях целесообразность использования горизонтальной фрагментации вполне очевидна. Однако в других случаях потребуется выполнение детального анализа приложений. Этот анализ должен включать проверку предикатов (или условий) 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 задавать диапазон значений.

Стратегия определения типа фрагментации предполагает поиск набора **минимальных** (т.е. полных и релевантных) предикатов, которые можно будет использовать как основу для построения схемы фрагментации (Ceri et al., 1982). 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

Вертикальный фрагмент	Выделенный по вертикали фрагмент отношения, 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 (см. раздел 3.4.1). Для заданного отношения R вертикальный фрагмент может быть вычислен с помощью следующей формулы:

$\Pi_{a1,\ldots,an}(R)$

Здесь a1, ... , an представляют собой атрибуты отношения R.

Пример 19.3. Вертикальная фрагментация

В компании *DreamHome* приложение, печатающее платежные ведомости, для каждого из работников компании использует атрибуты личного номера работника Sno, а также атрибуты Position (Должность), Sex (Пол), DOB (Дата рождения), Salary (Заработная плата) и NIN (Личный страховой номер). Ведомость, выдаваемая для отдела кадров, содержит атрибуты Sno, FName (Имя), LName (Фамилия), Address (Адрес), Tel_No (Номер телефона) и Bno (Номер отделения компании). Исходя из этих сведений, вертикальная фрагментация отношения Staff может быть выполнена с помощью следующих определений:

S_1: $\Pi_{sno,position,\ sex,\ dob,\ salary,\ nin}(Staff)$

S_2: $\Pi_{sno,\ fname,\ lname,\ address,\ tel\ no,\ bno}(Staff)$

С помощью этих формул будут созданы два фрагмента, содержимое которых представлено в табл. 19.6 и 19.7. Обратите внимание, что оба фрагмента содержат первичный ключ — атрибут Sno, — что позволяет при необходимости реконструировать исходное отношение. Преимущество вертикальной фрагментации состоит в том, что отдельные фрагменты могут размещаться на тех сайтах, на которых они используются. Это дополнительно оказывает положительное влияние на производительность системы, поскольку размеры каждого из фрагментов меньше размеров исходной таблицы. Приведенная схема фрагментации удовлетворяет правилам корректности.

- *Полнота.* Каждый атрибут отношения Staff присутствует либо во фрагменте S_1, либо во фрагменте S_2.
- *Восстановимость.* Исходное отношение Staff может быть реконструировано из отдельных фрагментов с помощью операции естественного соединения:

  $S_1 \bowtie S_2 = Staff$
- *Непересекаемость.* Содержимое отдельных фрагментов не пересекается, за исключением атрибута первичного ключа Sno, необходимого для реконструкции исходного отношения.

Таблица 19.6. Вертикальный фрагмент S_1 отношения Staff

Sno	Position	Sex	DOB	Salary	NIN
SL21	Manager	M	1-Oct-45	30000	WK442011B
SG37	Snr Asst	F	10-Nov-60	12000	WL432514C
SG14	Deputy	M	24-Mar-58	18000	WL220658D
SA9	Assistant	F	19-Feb-70	9000	WM532187D
SG5	Manager	F	3-Jun-40	24000	WK588932E
SL41	Assistant	F	13-Jun-65	9000	WA290573K

Таблица 19.7. Вертикальный фрагмент S_2 отношения Staff

Sno	FName	LName	Address	Tel_No	Bno
SL21	John	White	19 Taylor St, Cranford, London	0171-884-5112	B5
SG37	Ann	Beech	81 George St, Glasgow PA1 2JR	0141-848-3345	B3
SG14	David	Ford	63 Ashby St, Partick, Glasgow G11	0141-339-2177	B3
SA9	Mary	Howe	2 Elm Pl, Aberdeen AB2 3SU		B7
SG5	Susan	Brand	5 Gt Western Rd Glasgow G12	0141-334-2001	B3
SL41	Julie	Lee	28 Malvern St, Kilburn NW2	0181-554-3541	B5

Вертикальные фрагменты определяются путем установки **родственности** одного атрибута по отношению к другому. Один из способов решить эту задачу состоит в создании матрицы, содержащей количество обращений с выборкой каждой из пар атрибутов. Например, транзакция, которая осуществляет доступ к атрибутам a_1, a_2 и a_4 отношения R, состоящего из набора атрибутов (a_1, a_2, a_3, a_4), может быть представлена следующей матрицей:

	a_1	a_2	a_3	a_4
a_1		1	0	1
a_2			0	1
a_3				0
a_4				

Эта матрица является треугольной, поскольку диагональ ее не заполняется, а нижняя часть является зеркальным отражением верхней части. Единицы в матрице означают наличие доступа с обращением к соответствующей паре атрибутов и, в конечном счете, должны быть заменены числами, отражающими частоту выполнения транзакции. Подобная матрица составляется для каждой транзакции, после чего строится общая матрица, содержащая суммы всех показателей доступа к каждой из пар атрибутов. Пары атрибутов с высоким показателем родственности должны присутствовать в одном и том же вертикальном фрагменте. Пары с невысоким показателем родственности могут быть разнесены в разные вертикальные фрагменты. Очевидно, что обработка сведений об отдельных атрибутах для всех важнейших транзакций может потребовать немало времени и вычислений. Следовательно, если заранее известно о родственности определенных атрибутов, может оказаться целесообразным обработать сведения сразу о группах атрибутов.

Подобный подход носит название расщепления (splitting) и впервые был предложен группой разработчиков в 1984 году (Navathe et al., 1984). Он позволяет выделить набор неперекрывающихся фрагментов, которые гарантированно будут отвечать определенному выше правилу непересекаемости. Фактически требование непересекаемости касается только атрибутов, не входящих в первичный ключ отношения. Атрибуты первичного ключа должны присутствовать в каждом из выделенных вертикальных фрагментов, а потому могут быть исключены из анализа. Дополнительную информацию, касающуюся данного метода, заинтересованный читатель может найти в работе Оцзу и Валдуриса (Ozsu and Valduries, 1997).

Смешанная фрагментация

В некоторых случаях применения только лишь горизонтальной и вертикальной фрагментации элементов схемы базы данных оказывается недостаточно для адекватного распределения данных между приложениями. В этом случае приходится прибегать к **смешанной** (или **гибридной**) фрагментации.

Смешанный фрагмент	Образуется либо посредством дополнительной вертикальной фрагментации созданных ранее горизонтальных фрагментов, либо за счет вторичной горизонтальной фрагментации предварительно определенных вертикальных фрагментов.

Смешанная фрагментация определяется с помощью операций выборки и проекции реляционной алгебры. Если имеется некоторое отношение R, то смешанный фрагмент может быть определен по формулам

$\sigma_p(\Pi_{a1, \ldots, an}(R))$ или

$\Pi_{a1, \ldots, an}(\sigma_p(R))$

Здесь p является предикатом, построенным на использовании одного или больше атрибутов отношения R, обозначенных в формулах символами $a_1, \ldots, a_n$.

Пример 19.4. Смешанная фрагментация

В примере 19.3 мы выполнили вертикальную фрагментацию отношения Staff для приложений печати платежной ведомости и некоторого документа отдела кадров. Разбиение было определено с помощью следующих формул:

S_1: $\Pi_{sno, position, sex, dob, salary, nin}(Staff)$

S_2: $\Pi_{sno, fname, lname, address, tel\ no, bno}(Staff)$

Теперь можно выполнить дополнительную горизонтальную фрагментацию фрагмента S_2 по атрибуту номера отделения компании bno:

S_{21}: $\sigma_{bno='B3'}(S_2)$

S_{22}: $\sigma_{bno='B5'}(S_2)$

S_{23}: $\sigma_{bno='B7'}(S_2)$

В результате будут созданы три фрагмента, первый из которых включает кортежи с номером отделения, равным 'B3' (табл. 19.9), второй — с номером отделения, равным 'B5' (табл. 19.10), а в третий вошли кортежи с номером отделения, равным 'B7' (табл. 19.11). Содержимое фрагмента S_1 представлено в табл. 19.8. Полученная схема фрагментации удовлетворяет правилам корректности.

- *Полнота.* Каждый из атрибутов исходного отношения Staff присутствует либо во фрагменте S_1, либо во фрагменте S_2; каждый кортеж (в виде отдельных частей) исходного отношения присутствует во фрагменте S_1, а также в отношении S_{21}, S_{22} или S_{23}.
- *Восстановимость.* Исходное отношение Staff может быть восстановлено из полученных фрагментов путем выполнения операций объединения и естественного соединения с применением следующей формулы:

$$S_1 \bowtie (S_{21} \cup S_{22} \cup S_{23}) = Staff$$

- *Непересекаемость.* Полученные фрагменты не пересекаются, поскольку не существует работника, зачисленного более чем в одно отделение компании, а фрагменты S_1 и S_2 содержат различные атрибуты, за исключением обязательного атрибута первичного ключа.

Таблица 19.8. Смешанная фрагментация отношения Staff. Фрагмент S_1

Sno	Position	Sex	DOB	Salary	NIN
SL21	Manager	M	1-Oct-45	30000	WK442011B
SG37	Snr Asst	F	10-Nov-60	12000	WL432514C
SG14	Deputy	M	24-Mar-58	18000	WL220658D
SA9	Assistant	F	19-Feb-70	9000	WM532187D
SG5	Manager	F	3-Jun-40	24000	WK588932E
SL41	Assistant	F	13-Jun-65	9000	WA290573K

Таблица 19.9. Смешанная фрагментация отношения Staff. Фрагмент S_{21}

Sno	FName	LName	Address	Tel_No	Bno
SG37	Ann	Beech	81 George St, Glasgow PA1 2JR	0141-848-3345	B3
SG14	David	Ford	63 Ashby St, Partick, Glasgow G11	0141-339-2177	B3
SG5	Susan	Brand	5 Gt Western Rd Glasgow G12	0141-334-2001	B3

Таблица 19.10. Смешанная фрагментация отношения Staff. Фрагмент S_{22}

Sno	FName	LName	Address	Tel_No	Bno
SL21	John	White	19 Taylor St, Cranford, London	0171-884-5112	B5
SL41	Julie	Lee	28 Malvern St, Kilburn NW2	0181-554-3541	B5

Таблица 19.11. Смешанная фрагментация отношения Staff. Фрагмент S_{23}

Sno	FName	LName	Address	Tel_No	Bno
SA9	Mary	Howe	2 Elm Pl, Aberdeen AB2 3SU		B7

Производная горизонтальная фрагментация

Некоторые приложения включают операции соединения двух или больше отношений. Если отношения сохраняются в различных местах, то выполнение их соединения создаст очень большую дополнительную нагрузку на систему. В подобных случаях более приемлемым решением будет размещение соединяемых отношений или их фрагментов в одном и том же месте. Данная цель может быть достигнута за счет применения производной горизонтальной фрагментации.

Производный фрагмент	Горизонтальный фрагмент отношения, созданный на основе горизонтального фрагмента родительского отношения.

Термин *"дочернее"* мы будем использовать для ссылок на отношение, содержащее внешний ключ, а термин *"родительское"* — для ссылок на отношение с соответствующим первичным ключом. Определение производных фрагментов осуществляется с помощью операции *полусоединения* реляционной алгебры (см. раздел 3.4.1). Если заданы дочернее отношение R и родительское отношение S, то производный фрагмент отношения R может быть определен следующим образом:

$$R_i = R \ltimes_F S_i, \qquad 1 \le i \le w$$

Здесь значение w — это количество горизонтальных фрагментов, определенных для отношения S, а параметр F задает атрибут, по которому выполняется соединение.

Пример 19.5. Производная горизонтальная фрагментация

Допустим, что существует приложение, в котором выполняется соединение отношений Staff и Property_for_Rent. Кроме того, предположим, что отношение Staff разбито на три горизонтальных фрагмента в соответствии со значением атрибута номера отделения компании Bno. Каждый полученный фрагмент ведется на соответствующем сайте локально.

$$S_3 = \sigma_{bno='B3'}(Staff)$$

$$S_4 = \sigma_{bno='B5'}(Staff)$$

$$S_5 = \sigma_{bno='B7'}(Staff)$$

Следовательно, имеет смысл хранить сведения об объектах недвижимости с применением того же способа фрагментации. Данная цель достигается за счет применения метода производной фрагментации для горизонтального разбиения отношения Property_for_Rent в соответствии с номером отделения компании:

$$P_i = Property\_for\_Rent \triangleright_{bno} S_i, \qquad 3 \le i \le 5$$

В результате будут созданы три фрагмента. Первый из них будет содержать сведения об объектах недвижимости, обслуживаемых в отделении компании с номером 'B3'

(табл. 19.12), второй фрагмент описывает объекты недвижимости, обслуживаемые отделением с номером 'B5' (табл. 19.13), а третий — отделением с номером 'B7' (табл. 19.14). Несложно доказать, что полученная схема фрагментации отвечает правилам корректности. Мы оставляем это доказательство (в качестве упражнения) для читателей.

Таблица 19.12. Производная фрагментация отношения Property_for_Rent по отношению Staff. Фрагмент P_3

Pno	Sreet	Area	City	Pcode	Type	Rooms	Rent	Ono	Sno
PG4	6 Lawrence St	Partick	Glasgow	G11 9QX	Flat	3	350	CO40	SG14
PG36	2 Manor Rd		Glasgow	G32 4QX	Flat	3	375	CO93	SG37
PG21	18 Dale Rd	Hyndland	Glasgow	G12	House	5	600	CO87	SG37
PG16	5 Novar Dr	Hyndland	Glasgow	G12 9AX	Flat	4	450	CO93	SG14

Таблица 19.13. Производная фрагментация отношения Property_for_Rent по отношению Staff. Фрагмент P_4

Pno	Sreet	Area	City	Pcode	Type	Rooms	Rent	Ono	Sno
PL94	6 Argyll St	Kilburn	London	NW2	Flat	4	400	CO87	SL41

Таблица 19.14. Производная фрагментация отношения Property_for_Rent по отношению Staff. Фрагмент P_5

Pno	Sreet	Area	City	Pcode	Type	Rooms	Rent	Ono	Sno
PA14	16 Holhead	Dee	Aberdeen	AB7 5SU	House	6	650	CO46	SA9

Если отношение включает больше одного внешнего ключа, то может потребоваться выбрать в качестве родительского только одно из связанных отношений. Выбор может быть сделан в соответствии с чаще всего используемым типом фрагментации или с целью достижения оптимальных характеристик соединения — например, соединения, которое включает более мелкие фрагменты или соединения, выполняемого с большей степенью распараллеливания.

Отказ от фрагментации

Последний вариант возможной стратегии состоит в отказе от фрагментации отношения. Например, отношение Branch содержит небольшое количество кортежей, которые относительно редко обновляются. Вместо того чтобы попытаться выполнить горизонтальную фрагментацию этого отношения (например, по номеру отделения компании), имеет смысл оставить это отношение нефрагментированным и просто разместить на каждом из сайтов его реплицируемые копии. Это первый этап типовой процедуры определения схемы фрагментации (поиск отношений, которые не нуждаются в фрагментации). Затем следует проанализировать отношения, расположенные на единичной стороне связей типа "один ко многим", и подобрать для них оптимальные схемы фрагментации. На последнем этапе анализируются отношения, расположенные на множественной стороне тех же связей. Именно они чаще всего являются кандидатами для применения производной фрагментации.

19.5. Обеспечение прозрачности в РСУБД

В определении РСУБД, приведенном в разделе 19.1.1, утверждается, что система должна обеспечить прозрачность распределенного хранения данных для конечного пользователя. Под прозрачностью понимается сокрытие от пользователей деталей реализации системы. Например, в централизованной СУБД обеспечение независимости программ от данных также можно рассматривать как одну из форм прозрачности — в данном случае от пользователя скрываются изменения, происходящие в определении и организации хранения данных. Распределенные СУБД могут обеспечивать различные уровни прозрачности. Однако в любом случае преследуется одна и та же цель: сделать работу с распределенной базой данных совершенно аналогичной работе с обычной централизованной СУБД. Все виды прозрачности, которые мы будем обсуждать ниже, очень редко можно найти в какой-либо одной СУРБД. Мы выделим четыре основных типа прозрачности, которые могут иметь место в системе с распределенной базой данных.

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

Прежде чем приступить к обсуждению каждого из этих типов прозрачности, следует отметить, что полная прозрачность не всегда принимается как одна из целевых установок. Например, Грей в одной из своих работ (Gray, 1989) утверждает, что полная прозрачность превращает управление распределенными данными в чрезвычайно трудную задачу. Кроме того, он указывает, что приложения, написанные с учетом полной прозрачности доступа в географически распределенной базе данных, обычно характеризуются недостаточными управляемостью, модульностью и производительностью обработки сообщений.

19.5.1. Прозрачность распределенности

Прозрачность распределенности базы данных позволяет конечным пользователям воспринимать базу данных как единое логическое целое. Если СУРБД обеспечивает прозрачность распределенности, то пользователю не требуется каких-либо знаний о фрагментации данных (**прозрачность фрагментации**) или их размещении (**прозрачность расположения**).

Если пользователю необходимо иметь сведения о фрагментации данных и расположении фрагментов, то этот тип прозрачности мы будем называть **прозрачностью локального отображения**. Ниже мы подробно рассмотрим все упомянутые типы прозрачности. Для иллюстрации обсуждаемых концепций будет использоваться отношение Staff, фрагментированное так, как описано в примере 19.4, а именно:

S_1:	$\Pi_{sno,\ position,\ sex,\ dob,\ salary,\ nin}(Staff)$	фрагмент расположен на сайте 5
S_2:	$\Pi_{sno,\ fname,\ lname,\ address,\ tel\ no,\ bno}(Staff)$	
S_{21}:	$\sigma_{bno='B3'}(S2)$	фрагмент расположен на сайте 3
S_{22}:	$\sigma_{bno='B5'}(S2)$	фрагмент расположен на сайте 5
S_{23}:	$\sigma_{bno='B7'}(S2)$	фрагмент расположен на сайте 7

Прозрачность фрагментации

Прозрачность фрагментации является самым высоким уровнем прозрачности распределенности. Если СУРБД 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 вы-

борки сведений обо всех руководителях отделений (у них атрибут Position имеет значение 'Manager'), при наличии в системе прозрачности фрагментации, можно воспользоваться следующим SQL-оператором:

```
SELECT fname, lname
FROM STAFF
WHERE position = 'Manager';
```

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

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 fname, lname
FROM S21
WHERE sno IN (SELECT sno FROM S1 WHERE position = 'Manager') UNION
SELECT fname, lname
FROM S22
WHERE sno IN (SELECT sno FROM S1 WHERE position = 'Manager') UNION
SELECT fname, lname
FROM S23
WHERE sno IN (SELECT sno FROM S1 WHERE position = 'Manager');
```

В этом случае в запросе необходимо указывать имена используемых фрагментов. Кроме того, дополнительно необходимо воспользоваться операциями соединения (или подзапросами), поскольку атрибуты Position и FName/LName находятся в разных вертикальных фрагментах. Основное преимущество прозрачности расположения состоит в том, что база данных может быть подвергнута физической реорганизации и это не окажет никакого влияния на программы приложений, осуществляющих к ней доступ.

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. Однако 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 fname, lname
FROM S21 AT SITE 3
WHERE sno IN (SELECT sno FROM S1 AT SITE 5 WHERE position='Manager')
UNION
SELECT fname, lname
FROM S22 AT SITE 5
WHERE sno IN (SELECT sno FROM S1 AT SITE 5 WHERE position='Manager')
UNION
```

```
SELECT fname, lname
FROM S23 AT SITE 7
WHERE sno IN (SELECT sno FROM S1 AT SITE 5 WHERE position='Manager');
```

Из соображений наглядности, мы в данном случае дополнили язык SQL 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, СУРБД должна гарантировать, что никакие два сайта 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 центральный сайт превращается в узкое место всей системы);
- снижение доступности — если центральный сайт по какой-либо причине станет недоступным, все остальные сайты системы не смогут создавать никаких новых объектов базы данных.

Альтернативное решение состоит в использовании префиксов, помещаемых в имена объектов в качестве идентификатора сайта, создавшего этот объект. Например, отношение Branch, созданное на сайте S_1, могло бы получить имя S1.BRANCH. Аналогичным образом, необходимо иметь возможность идентифицировать каждый фрагмент и каждую его копию. Поэтому второй копии третьего фрагмента отношения Branch, созданного на сайте S_1, можно было бы присвоить имя S1.BRANCH.F3.C2. Однако подобный подход приводит к утрате прозрачности распределенности.

Подход, который позволяет преодолеть недостатки, свойственные обоим упомянутым методам, состоит в использовании **алиасов**, создаваемых для каждого из объектов базы данных. В результате объект S1.BRANCH.F3.C2 пользователям сайта S_1 может быть известен под именем local_branch. Задача преобразования алиаса в истинное имя соответствующего объекта базы данных возлагается на СУРБД.

19.5.2. Прозрачность транзакций

Прозрачность транзакций в среде распределенных СУБД означает, что при выполнении любых распределенных транзакций гарантируется сохранение целостности и согласованности распределенной базы данных. **Распределенная транзакция** осуществляет доступ к данным, сохраняемым более чем в одном местоположении. Каждая из транзакций разделяется на несколько **субтранзакций** — по одной для каждого сайта, к данным которого осуществляется доступ. На удаленных сайтах субтранзакции представляются **агентами**.

Пример 19.6. Распределенная транзакция

Рассмотрим выполнение транзакции T, выполняющей распечатку имен всего персонала компании, при использовании схемы фрагментации, определенной выше в виде фрагментов S_1, S_2, S_{21}, S_{22} и S_{23}. Транзакция будет включать три субтранзакции T_{S3}, T_{S5} и T_{S7}, представленные агентами на сайтах 3, 5 и 7 соответственно. Ка-

ждая из субтранзакций печатает имена работников локального отделения компании. График распределенной транзакции показан в табл. 19.15. Обратите внимание на естественную параллельность, свойственную системе, — каждая из субтранзакций выполняется на своем сайте параллельно с остальными.

Таблица 19.15. Пример распределенной транзакции

Время	Транзакция T_{S_3}	Транзакция T_{S_5}	Транзакция T_{S_7}
t_1	begin_transaction	begin_transaction	begin_transaction
t_2	read(fname, lname)	read(fname, lname)	read(fname, lname)
t_3	print(fname, lname)	print(fname, lname)	print(fname, lname)
t_4	end_transaction	end_transaction	end_transaction

Неделимость остается фундаментальной концепцией понятия транзакции и в случае распределенных транзакций, однако дополнительно СУРБД должна гарантировать неделимость и каждой из ее субтранзакций (см. раздел 17.1.1). Следовательно, СУРБД должна гарантировать не только синхронизацию субтранзакций с другими локальными транзакциями, выполняющимися параллельно с ними на одном сайте, но и обеспечить синхронизацию субтранзакций с глобальными транзакциями, выполняющимися одновременно с ними на этом и других сайтах системы. Прозрачность транзакций в распределенных СУБД дополнительно усложняется за счет наличия фрагментации, распределения данных и использования репликации. Мы рассмотрим два дополнительных аспекта прозрачности транзакций, таких как **прозрачность параллельности** и **прозрачность отказов**.

Прозрачность параллельности

Прозрачность параллельности обеспечивается СУБРД в том случае, если результаты всех параллельно выполняемых транзакций (как распределенных, так и нераспределенных) генерируются независимо и являются логически согласующимися с результатами, которые были бы получены в том случае, если бы все эти транзакции выполнялись последовательно в некотором произвольном порядке, по одной в каждый момент времени. Существуют некоторые фундаментальные принципы, которые мы уже обсуждали при рассмотрении централизованных СУБД в разделе 17.2.2. Однако в случае распределенных СУБД имеют место дополнительные усложнения, связанные с необходимостью гарантировать, что как глобальные, так и локальные транзакции не могут оказывать влияния друг на друга. Кроме того, СУРБД должны гарантировать согласованность всех субтранзакций каждой глобальной транзакции.

Наличие в системе репликации еще более усложняет проблему организации параллельной обработки в системе. Если одна из копий реплицируемых данных подвергается обновлению, сведения об этом в конечном счете должны быть представлены в каждой из существующих копий. В данном случае наиболее очевидная стратегия — сделать распространение сведений об изменении частью исходной транзакции, оформив его как еще одну атомарную операцию. Однако, если один из содержащих копию измененных данных сайтов окажется в момент внесения изменения недоступным из-за отказа на самом сайте или в канале связи, то выполнение транзакции будет отложено до тех пор, пока этот сайт вновь не станет доступным. Если существует большое количество копий данных, то вероятность успешного завершения транзакции уменьшается в экспоненциальной зависимости от их числа. Альтернативной стратегией является ограничение распространения сведений об изменении только теми сайтами, которые в данный момент доступны. На остальные сайты сведения об изменении поступят, как только они вновь станут доступными. Дополнительной стратегией могла бы быть выдача разрешения обновлять копии асинхронно, через некоторое время после внесения исходного обновления. Задержка в восстановлении целостности может варьироваться от нескольких секунд до нескольких часов. Подробное обсуждение корректных методов организации распределенной параллельности мы проведем в следующей главе.

Прозрачность отказов

В разделе 17.3.2 утверждалось, что любая централизованная СУБД должна включать механизм восстановления, который в подверженной различным сбоям и отказам среде будет гарантировать **атомарность** выполнения транзакций — либо все операции транзакции будут успешно завершены, либо ни одна из них. Более того, если результаты выполнения транзакции были зафиксированы, внесенные ею изменения приобретают **длительный** (или постоянный) характер. Также были рассмотрены различные типы отказов, которые могут иметь место в централизованных СУБД: сбои системы, отказ носителей, ошибки в программах, небрежность персонала, стихийные бедствия и действия злоумышленников. В распределенной среде СУРБД должна дополнительно учитывать следующее:

- возможность потери сообщения;
- возможность отказа линии связи;
- аварийный останов одного из сайтов;
- расчленение сетевых соединений.

СУРБД должна гарантировать атомарность глобальных транзакций, а это означает, что все ее субтранзакции будут либо зафиксированы, либо отменены. Следовательно, СУРБД должна синхронизировать выполнение глобальной транзакции таким образом, чтобы иметь гарантии, что все ее субтранзакции были успешно завершены до того, как началась финальная операция фиксации результатов всей глобальной транзакции. Например, рассмотрим глобальную транзакцию, которая выполняет обновление данных на двух сайтах, S_1 и S_2. Пусть субтранзакция на сайте S_1 завершается успешно и фиксирует свои результаты, однако субтранзакция на сайте S_2 не может успешно завершить свое выполнение и производит откат внесенных изменений для сохранения целостности локальной базы данных. В результате распределенная транзакция оказывается в несогласованном состоянии, поскольку из-за свойства длительности транзакций результаты выполнения транзакции S_1 отменить уже нельзя. Обсуждение корректных методов восстановления распределенных баз данных будет предложено вашему вниманию в следующей главе.

19.5.3. Прозрачность выполнения

Прозрачность выполнения требует, чтобы работа в среде СУРБД выполнялась точно так же, как и в среде централизованной СУБД. В распределенной среде работа системы не должна демонстрировать никакого снижения производительности, связанного с ее распределенной архитектурой, например с присутствием медленных сетевых соединений. Прозрачность выполнения также требует, чтобы СУРБД была способна находить наиболее эффективные стратегии выполнения запросов.

В централизованной СУБД обработчик запросов должен оценивать каждый запрос на доступ к данным и находить оптимальную стратегию его выполнения, представляющую собой упорядоченную последовательность операций с базой данных. В распределенной среде обработчик распределенных запросов отображает запрос на доступ к данным в упорядоченную последовательность операций локальных баз данных. Дополнительная сложность возникает из-за необходимости учитывать наличие фрагментации, репликации и определенной схемы размещения данных. Обработчик распределенных запросов должен выяснить:

- к какому фрагменту следует обратиться;
- какую копию фрагмента использовать, если его данные реплицируются;
- какое из местоположений должно использоваться.

Обработчик распределенных запросов вырабатывает стратегию выполнения, которая является оптимальной с точки зрения некоторой стоимостной функции. Обычно распределенные запросы оцениваются по таким показателям:

- время доступа, включающее физический доступ к данным на диске;
- время работы центрального процессора, затрачиваемое на обработку данных в первичной памяти;
- время, необходимое для передачи данных по сетевым соединениям.

Первые два фактора аналогичны тем, которые учитываются в централизованных системах. Однако в распределенной среде СУРБД необходимо учитывать и затраты на передачу данных, которые во многих случаях среди прочих затрат оказываются доминирующими. Последнее замечание будет особенно справедливо в случае использования медленных сетевых соединений, характерных для глобальных сетей. Скорость передачи данных в таких каналах может составлять лишь несколько килобайт в секунду. В подобных ситуациях при оптимизации можно игнорировать затраты на ввод/вывод и использование ЦП. Однако некоторые сетевые соединения имеют скорость передачи данных, сравнимую со скоростью доступа к дискам (например, локальные сетевые соединения). В этом случае при оптимизации должны учитываться все три показателя затрат.

Один из подходов к оптимизации запросов предполагает минимизацию общих затрат времени, связанных с выполнением запроса (Sacco and Yao, 1982). Другой подход предусматривает минимизацию времени реакции системы на запрос. При этом основная задача оптимизатора запросов состоит в максимальном распараллеливании выполнения операций (Epstein et al., 1978). В некоторых случаях время реакции системы на запрос может быть существенно меньше общих затрат времени на его выполнение. Ниже приводится пример, взятый из работы Ротни и Гудмена (Rothnie and Goodman, 1977). Он иллюстрирует возможную ширину диапазона времени реакции системы на запрос в зависимости от выбора различных, но вполне допустимых стратегий его выполнения.

Пример 19.7. Обработка распределенного запроса

Рассмотрим упрощенный вариант реляционной схемы приложения *DreamHome*, включающий следующих три отношения:

Property(Pno, City)	10 000 записей, хранимых в Лондоне
Renter(Rno, Max_Price)	100 000 записей, хранимых в Глазго
Viewing(Pno, Rno)	1 000 000 записей, хранимых в Лондоне

Чтобы получить список объектов недвижимости в Абердине, которые были осмотрены потенциальными покупателями, согласными приобрести объекты недвижимости дороже 200 000 фунтов стерлингов, можно воспользоваться следующим SQL-запросом:

```
SELECT p.pno
FROM property p INNER JOIN
    (renter r INNER JOIN viewing v ON r.rno = v.rno)
    ON p.pno = v.pno
WHERE p.city = 'Aberdeen' AND r.max price > 200000;
```

Для простоты предположим, что каждый кортеж в отношениях имеет длину, равную 100 символам, что существует только 10 покупателей, согласных приобрести недвижимость по цене больше 200 000 фунтов стерлингов, и что в городе Абердин было проведено 100 000 осмотров объектов недвижимости. Примем также, что время выполнения вычислений несущественно по сравнению с временем передачи данных, а скорость передачи данных по каналам связи составляет 10 000 символов в секунду, причем на каждое отправляемое между двумя сайтами сообщение приходится задержка доступа, равная 1 секунде.

Ротни (Rothnie) проанализировал шесть возможных стратегий выполнения этого запроса и для каждого из них вычислил соответствующее время реакции системы. Для определения времени передачи данных применялся алгоритм, описанный в разделе 19.2.

Стратегия 1. Переслать отношение Renter в Лондон и выполнить там обработку запроса:

Время = 1+(100 000 * 100/10 000) ≈ 16,7 мин.

Стратегия 2. Переслать отношения Property и Viewing в Глазго и выполнить там обработку запроса:

Время = 2+[(1 000 000 + 10 000) * 100/10 000] ≈ 28 ч.

Стратегия 3. Соединить отношения Property и Viewing в Лондоне, выбрать кортежи для объектов недвижимости, расположенных в Абердине, а затем для каждого из отобранных кортежей проверить в Глазго, установил ли данный клиент значение потолка допустимой стоимости недвижимости Max_Price > 200 000 фунтов стерлингов. Проверка каждого кортежа предполагает отправку двух сообщений: запроса и ответа.

Время = 100 000 * (1+100/10 000) + 100 000 * 1 ≈ 2,3 дня.

Стратегия 4. Выбрать в Глазго кортежи клиентов, установивших значение Max_Price > 200 000 фунтов стерлингов, после чего для каждого из них проверить в Лондоне, осматривал ли этот клиент объекты недвижимости в Абердине. И в этом случае каждая проверка включает отправку двух сообщений.

Время = 10*(1+100/10 000) + 10* 1 ≈ 20 с.

Стратегия 5. Соединить отношения Property и Viewing в Лондоне, выбрать сведения об осмотрах объектов в Абердине, выполнить проекцию результирующей таблицы по атрибутам Pno и Rno, после чего переслать полученный результат в Глазго для отбора кортежей по условию Max_Price > 200 000 фунтов стерлингов. Для упрощения предположим, что длина кортежа после операции проекции также равна 100 символам.

Время = 1 + (100 000 * 100/10 000) ≈ 16,7 мин.

Стратегия 6. Выбрать клиентов со значением атрибута Max_Price > 200 000 фунтов стерлингов в Глазго и переслать результирующую таблицу в Лондон для отбора сведений об осмотрах объектов недвижимости в г. Абердин:

Время = 1 + (10 * 100/10 000) ≈ 1 с.

Таблица 19.16. Сравнение результатов применения различных стратегий для обработки одного и того же распределенного запроса

№	Стратегия	Время
1	Переслать отношение Renter в Лондон и выполнить там обработку запроса	16,7 мин
2	Переслать отношения Property и Viewing в Глазго и выполнить там обработку запроса	28 ч
3	Соединить отношения Property и Viewing в Лондоне, выбрать кортежи для объектов недвижимости в Абердине, а затем для каждого из отобранных кортежей проверить в Глазго, установил ли данный клиент значение потолка допустимой стоимости недвижимости Max_Price > 200 000 фунтов стерлингов	2,3 дней
4	Выбрать в городе Глазго кортежи клиентов, установивших значение Max_Price > 200 000 фунтов стерлингов, после чего для каждого из них проверить в Лондоне, осматривал ли этот клиент объекты недвижимости в Абердине	20 с

Окончание табл. 19.16

№	Стратегия	Время
5	Соединить отношения Property и Viewing в Лондоне, выбрать сведения об осмотрах объектов в Абердине, выполнить проекцию результирующей таблицы по атрибутам Pno и Rno, после чего переслать полученный результат в Глазго для отбора кортежей по условию Max_Price > 200 000 фунтов стерлингов	16,7 мин
6	Выбрать клиентов со значением атрибута Max_Price > 200 000 фунтов стерлингов в Глазго и переслать результирующую таблицу в Лондон для отбора сведений об осмотрах объектов недвижимости в Абердине	1 с

Описание стратегий и время получения результата для каждой из них показаны в табл. 19.16. Как можно видеть, время ответа системы изменяется в диапазоне от 1 секунды до 2,3 дней, хотя каждая из предложенных стратегий является вполне корректной и позволяет получить ответ на данный запрос! Вполне очевидно, что, если для выполнения запроса будет выбрана неподходящая стратегия, это может оказать крайне негативное влияние на уровень производительности системы. Обработка распределенных запросов будет подробно обсуждаться в разделе 20.7.

19.5.4. Прозрачность использования СУБД

Прозрачность использования СУБД позволяет скрыть от пользователя СУРБД тот факт, что на разных сайтах могут функционировать различные локальные СУБД. Поэтому данный тип прозрачности применим только в случае гетерогенных распределенных систем. Как правило, это один из самых сложных в реализации типов прозрачности. Речь о проблемах, связанных с созданием гетерогенных систем, шла выше, в разделе 19.1.3.

19.5.5. Несколько слов в заключение

В начале этого раздела, посвященного обсуждению различных типов прозрачности в среде распределенных СУБД, отмечалось, что достижение полной прозрачности не всеми расценивается как желаемая цель. Как мы уже видели, концепция прозрачности не может быть отнесена к типу "все или ничего", поскольку может быть организована на нескольких различных уровнях. Каждый из уровней подразумевает наличие определенного набора соглашений между сайтами, входящими в систему. Например, при полной прозрачности сайты должны следовать общему соглашению по таким вопросам, как модель данных, интерпретация схем, представление данных и набор функциональности, предоставляемый на каждом из сайтов. С другой стороны спектра находятся непрозрачные системы, в которых единственным соглашением является формат обмена данными и набор функциональных возможностей, предоставляемых каждым сайтом в системе.

С точки зрения конечного пользователя полная прозрачность весьма желательна. Однако с точки зрения АБД локальной базы полностью прозрачный доступ может быть связан с трудностями осуществления контроля. Традиционный способ работы с представлениями, зачастую используемый для построения защитного механизма, в этой ситуации может оказаться недостаточно эффективным. Например, механизм работы с представлениями языка SQL позволяет ограничивать доступ к базовым таблицам или подмножеству данных базовой таблицы и разрешать его только конкретным пользователям. Однако он не позволяет также легко ограничивать доступ к данным на основе набора критериев, отличных от имени пользователя. В учебном примере *DreamHome* можно запретить конкретным пользователям удалять записи из таблицы Lease_Agreement, но невозможно простыми средствами защитить эту таблицу, разрешив удалять из нее только те строки, которые относятся к соглашениям об аренде, срок действия которых уже истек, все задолженности по платежам погашены арендатором, а сам объект недвижимости находится в удовлетворительном состоянии.

Проще обеспечить подобный тип функциональности с помощью процедур, которые будут запускаться удаленным пользователем. В этом случае локальные пользователи смогут работать с данными так же, как они работали прежде, при использовании стандартного механизма защиты СУБД. Однако удаленные пользователи смогут получать только такой тип доступа к данным, который реализован в предоставленном им наборе процедур, подобно тому, как это делается в объектно-ориентированных системах. Подобный тип *федеральной архитектуры* реализовать проще, чем обеспечить полную прозрачность системы, к тому же он предоставляет более высокий уровень локальной автономности.

19.6. Двенадцать правил Дейта для РСУБД

В последнем разделе данной главы мы перечислим двенадцать правил (или целей), которые были сформулированы Дейтом (Date, 1987) для типичной СУРБД. Основой для построения всех этих правил является то, что распределенная СУБД должна восприниматься конечным пользователем точно так же, как и привычная ему централизованная СУБД. Данные правила сходны с двенадцатью правилами Кодда для реляционных систем, представленными в разделе 3.6.

Правило 0. Основной принцип

> С точки зрения конечного пользователя распределенная система должна выглядеть в точности так, как и обычная, нераспределенная система.

Правило 1. Локальная автономность

Сайты в распределенной системе должны быть автономными. В данном контексте автономность означает следующее:

- локальные данные принадлежат локальным владельцам и сопровождаются локально;
- все локальные процессы остаются чисто локальными;
- все процессы на заданном сайте контролируются только этим сайтом.

Правило 2. Отсутствие опоры на центральный сайт

В системе не должно быть ни одного сайта, без которого система не сможет функционировать. Это означает, что в системе не должно существовать центральных серверов таких служб, как управление транзакциями, выявление взаимных блокировок, оптимизация запросов и управление глобальным системным каталогом.

Правило 3. Непрерывное функционирование

В идеале, в системе никогда не должна возникать потребность в плановом останове ее функционирования для выполнения таких операций, как:

- добавление или удаление сайта из системы;
- динамическое создание или удаление фрагментов из одного или нескольких сайтов.

Правило 4. Независимость от расположения

Независимость от расположения эквивалентна прозрачности расположения. Пользователь должен получать доступ к базе данных с любого из сайтов. Более того, пользователь должен получать доступ к любым данным так, как если бы они хранились на его сайте, независимо от того, где они физически сохраняются.

Правило 5. Независимость от фрагментации

Пользователь должен получать доступ к данным независимо от способа их фрагментации.

Правило 6. Независимость от репликации

Пользователь не должен нуждаться в сведениях о наличии репликации данных. Это значит, что пользователь не будет иметь средств для получения прямого доступа к конкретной копии элемента данных, а также не должен заботиться об обновлении всех имеющихся копий элемента данных.

Правило 7. Обработка распределенных запросов

Система должна поддерживать обработку запросов, ссылающихся на данные, расположенные на более чем одном сайте.

Правило 8. Обработка распределенных транзакций

Система должна поддерживать выполнение транзакций, как единицы восстановления. Система должна гарантировать, что выполнение как глобальных, так и локальных транзакций будет происходить с сохранением четырех основных свойств транзакций, а именно: атомарности, согласованности, изолированности и продолжительности.

Правило 9. Независимость от типа оборудования

СУРБД должна быть способна функционировать на оборудовании с различными вычислительными платформами.

Правило 10. Независимость от операционной системы

Прямым следствием предыдущего правила является требование, согласно которому СУРБД должна быть способна функционировать под управлением различных операционных систем.

Правило 11. Независимость от сетевой архитектуры

СУРБД должна быть способна функционировать в сетях с различной архитектурой и типами носителя.

Правило 12. Независимость от типа СУБД

СУРБД должна быть способна функционировать поверх различных локальных СУБД, возможно, с разным типом используемой модели данных. Другими словами, СУРБД должна поддерживать гетерогенность.

Последних четыре правила являются пока лишь идеальными. Поскольку их формулировка является слишком общей, и по причине недостаточности имеющихся стандартов на компьютерную и сетевую архитектуру, в обозримом будущем можно ожидать лишь частичного удовлетворения разработчиками указанных требований.

Резюме

- **Распределенная база данных** представляет собой набор логически связанных между собой разделяемых данных (и их описаний), которые физически распределены в некоторой компьютерной сети. **СУРБД** представляет собой программный комплекс, предназначенный для прозрачного управления распределенной базой данных.
- Распределенную СУБД не следует смешивать с **распределенной обработкой**, при которой доступ к централизованной СУБД одновременно предоставляется многим пользователям в компьютерной сети. СУРБД также отличается **от параллельной СУБД**, в которой локальная система управления базой данных функционирует с использованием нескольких процессоров и устройств вторичной памяти, что позволяет организовать параллельное выполнение операций (если это возможно) с целью повышения производительности системы.
- Преимущества СУРБД заключаются в том, что она позволяет отразить структуру организации и повышает возможности совместного использования уда-

ленных данных, повышает надежность, доступность и производительность системы, позволяет получить экономию средств, а также организовать модульное увеличение размеров системы.

- Все взаимодействия выполняются с помощью сетевых соединений, которые могут быть как локальными, так и глобальными. Локальные сетевые соединения устанавливаются на небольшие расстояния, но обеспечивают большую пропускную способность, чем глобальные.
- Аналогично тому, как централизованная СУБД должна предоставлять определенный набор стандартных функций, распределенная СУБД должна предоставлять расширенные возможности установки соединений, включать расширенную службу системного каталога, обеспечивать распределенную обработку запросов, поддерживать расширенные средства распараллеливания операций, а также иметь собственную службу восстановления.
- Каждое отношение может быть разделено на некоторое количество частей, называемых **фрагментами**. Фрагменты могут быть горизонтальными, вертикальными, смешанными и производными. Фрагменты **распределяются** на одном или нескольких сайтах. С целью улучшения доступности данных и повышения производительности системы для отдельных фрагментов может быть организована **репликация**.
- Определение и распределение фрагментов выполняются для достижения следующих целей: обеспечения локальности ссылок, повышения надежности и доступности данных, обеспечения приемлемого уровня производительности, достижения баланса между стоимостью и емкостью устройств вторичной памяти, а также минимизации расходов на передачу данных. Три основных правила корректности фрагментации включают требования полноты, восстановимости и непересекаемости.
- Существуют четыре стратегии распределения, определяющие способ размещения данных: **централизованное** (единственная централизованная база данных), **фрагментированное распределение** (каждый фрагмент размещается на одном из сайтов), **распределение с полной репликацией** (полная копия всей базы данных поддерживается на каждом сайте) и **распределение с выборочной репликацией** (комбинация первых трех способов).
- С точки зрения пользователя, СУРБД должна выглядеть точно так же, как и обычная централизованная СУБД, что достигается за счет обеспечения различных типов прозрачности. Благодаря **прозрачности распределения** пользователи не нуждаются в каких-либо сведениях о существующей в системе фрагментации/репликации. **Прозрачность транзакций** обеспечивает сохранение согласованности глобальной базы даже при наличии параллельного доступа к ней со стороны множества пользователей и наличия в системе различных отказов. **Прозрачность выполнения** позволяет системе эффективно обрабатывать запросы, включающие обращение к данным на нескольких сайтах. **Прозрачность использования СУБД** позволяет системе функционировать поверх установленных на отдельных сайтах локальных СУБД различного типа.

Вопросы

19.1. Поясните значение термина "СУРБД" и назовите причины создания подобных систем.

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

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

19.4. Назовите преимущества и недостатки, свойственные распределенным системам.

19.5. 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.

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

19.7. Дайте определение и укажите различия между альтернативными схемами фрагментации глобальных отношений. Поясните, как можно проверить корректность выполненных действий и получить гарантии того, что в процессе фрагментации в базу данных не было внесено семантических изменений.

19.8. Какие уровни прозрачности должны поддерживаться СУРБД? Обоснуйте ваш ответ.

19.9. СУРБД должна гарантировать, что на любой паре сайтов 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.

Упражнения

Крупная инженерная компания приняла решение распределить информацию, связанную с управлением выполняемыми в ней проектами, по всем регионам Великобритании. Существующая централизованная реляционная схема имеет следующее описание:

```
Employee     (NIN, First_Name, Last_Name, Address, Birth_Date, Sex, Salary,
              Tax_Code, Dept_No)
Department   (Dept_No, Dept_Name, Manager_NIN, Business_Area_No, Region_No)
Project      (Proj_No, Proj_Name, Contract_Price, Project_Manager_NIN, Dept_No)
Works_On     (NIN, Proj_No, Hours_Worked)
Business     (Business_Area_No, Business_Area_Name)
Region       (Region_No, Region_Name)
```

где
- Employee — отношение, содержащее сведения о работниках, первичный ключ — атрибут NIN
- Department — отношение, содержащее сведения о подразделениях компании, пер вичный ключ — атрибут Dept_No. Атрибут Manager_NIN определяет работника, являющегося руководителем подразделения. В каждом подразделении может быть только один руководитель.
- Project — отношение, содержащее сведения о выполняемых в компании проектах, первичный ключ — атрибут Proj_No. Руководитель проекта определяется атрибутом Project_Manager_NIN, а подразделение, в котором выполняется проект, — атрибутом Dept_No.
- Works_On — отношение, содержащее сведения о рабочих часах, отработанных работ никами по каждому проекту, первичный ключ — пара атрибутов (NIN, Proj_No).
- Business — отношение, содержащее наименования деловых областей, первичныйключ — Business_Area_No.
- Region — отношение, содержащее наименования регионов, первичный ключ — атрибут Region_No.

Подразделения компании группируются по регионам следующим образом:

Region 1: 'Scotland' (Шотландия);
Region 2: 'Wales' (Уэллс);
Region 3: 'England' (Англия).

Информация собирается по отдельным деловым областям, к которым относятся: 'Software Engineering' (Программирование), 'Mechanical Engineering' (Механика) и 'Electrical Engineering' (Электротехника). Проекты по программированию не выполняются в Уэллсе, а все подразделения, занимающиеся электротехникой, расположены в Англии. Для выполнения проектов набираются работники из местных подразделений компании.

Помимо регионального распределения данных, существует дополнительное требование — обеспечить доступ к данным о персонале либо по личным данным, либо по сведениям о выполняемой работе.

19.10. Подготовьте диаграмму "сущность–связь" (ER-диаграмму) для описанной выше системы.

19.11. Используя созданную в предыдущем упражнении ER-диаграмму, подготовьте проект распределенной базы данных для описанной выше системы, включающий следующее:

а) подходящую случаю схему фрагментации в системе;

б) при использовании первичной горизонтальной фрагментации подготовьте минимальный набор соответствующих предикатов;

в) приведите формулы реконструкции глобальных отношений из их фрагментов.

Обоснуйте любые сделанные вами при разработке проекта допущения.

19.12. Повторите упражнение 19.11 для учебного проекта *DreamHome*, описанного в разделе 1.7.

19.13. В разделе 19.5.1 (*при обсуждении понятия* прозрачности именования) для уникальной идентификации каждой реплики любого фрагмента было предложено использовать алиасы. Подготовьте эскизный проект реализации этого подхода для обеспечения прозрачности именования.

ГЛАВА 20

Распределенные СУБД – дополнительные концепции

В этой главе...

Влияние распределенности базы данных на компоненты управления транзакциями.

Расширение технологий централизованного управления параллельностью с целью применения в распределенной среде.

Обнаружение ситуации взаимной блокировки при взаимодействии нескольких сайтов.

Восстановление после отказа базы данных в распределенной среде:

- при использовании двухфазного протокола фиксации (2PC);
- при использовании трехфазного протокола фиксации (3PC).

Проблемы выявления нарушений и поддержания целостности данных в распределенной среде.

Стандарт обработки транзакций X/Open DTP.

Реализация механизмов репликации в различных коммерческих системах.

Оптимизация распределенных запросов.

Использование операций полусоединения в распределенной среде.

В предыдущей главе мы обсудили основные концепции построения систем управления распределенными базами данных (СУРБД) и выяснили, как можно расширить методологию концептуального и логического проектирования, предложенную в главах 7 и 8, для применения в подобных системах. С точки зрения пользователя предлагаемые СУРБД функциональные возможности выглядят весьма привлекательно. Однако с точки зрения реализации используемые для поддержки подобной функциональности протоколы и алгоритмы оказываются весьма сложными, что вызывает появление ряда серьезных проблем, в некоторых случаях способных даже перевесить те немалые преимущества, которые могут быть получены за счет развертывания СУРБД. В этой главе мы продолжим обсуждение технологии распределенных СУБД и познакомимся с тем, как обсуждавшиеся в главе 17 протоколы управления параллельностью, выявления взаимных блокировок и восстановления системы могут быть расширены с целью поддержки распределенности хранения данных и реализации механизмов репликации.

Структура этой главы

В разделе 20.1 кратко обсуждается назначение механизма управления распределенными транзакциями. В разделе 20.2 рассматривается, как может быть учтена распределенность хранения данных в определении понятия упорядоченности, приведенном в разделе 17.2.2, после чего обсуждаются методы расширения представленных в разделе 17.2 протоколов управления параллельностью с целью применения их в распределенной среде. В разделе 20.3 мы познакомимся с методами выявления состояния взаимной блокировки, сложность которых в распределенной среде существенно выше, а также обсудим протоколы, которые могут применяться в СУРБД для обнаружения взаимных блокировок. В разделе 20.4 рассматриваются отказы, которые могут иметь место в распределенной среде, а также протоколы, которые могут использоваться для получения гарантии атомарности распределенных транзакций и длительности результатов их выполнения. В разделе 20.5 кратко представлен протокол X/Open DTP, определяющий программный интерфейс для обработки транзакций. В разделе 20.6 мы обсудим методы реализации серверов репликации в существующих коммерческих СУБД как альтернативу использованию СУРБД. В заключение этой главы, в разделе 20.7, приведен краткий обзор методов оптимизации распределенных запросов. Все примеры в этой главе построены на использовании материала учебного проекта *DreamHome*, описание которого вы найдете в разделе 1.7.

20.1. Управление распределенными транзакциями

В разделе 19.2 было отмечено, что цели распределенной обработки транзакций аналогичны тем, которые преследуются при обработке транзакций в централизованных системах, хотя используемые для этого методы существенно сложнее, поскольку СУРБД должна гарантировать неделимость всей глобальной транзакции, а также каждой из входящих в ее состав субтранзакций. В разделе 17.1.2 были выделены четыре высокоуровневых модуля базы данных, предназначенных для обработки транзакций, управления параллельностью и обеспечения восстановления в централизованных СУБД. **Менеджер транзакций** координирует выполнение запускаемых прикладными программами транзакций и взаимодействует с **планировщиком**, ответственным за реализацию выбранной стратегии управления параллельностью в системе. Цель работы планировщика состоит в достижении максимального уровня параллельности в системе с одновременной гарантией отсутствия какого-либо влияния параллельно выполняющихся транзакций друг на друга, что необходимо для постоянного сохранения базы данных в согласованном состоянии. Если в процессе выполнения транзакции происходит отказ, **менеджер восстановления** гарантирует, что

база данных будет восстановлена в согласованном состоянии, которое она имела до начала выполнения транзакции. Менеджер восстановления также отвечает за восстановление базы данных в некотором согласованном состоянии и после отказов системы. **Менеджер буферов** обеспечивает передачу данных между дисковыми устройствами и оперативной памятью компьютера.

В распределенной СУБД все эти модули по-прежнему существуют в каждой локальной СУБД. Кроме того, на каждом сайте функционирует **менеджер глобальных транзакций** или **координатор транзакций**, координирующий выполнение глобальных и локальных транзакций, инициированных на данном сайте. Все взаимодействия между сайтами осуществляются через **компонент передачи данных** (менеджеры транзакций на различных сайтах не взаимодействуют друг с другом напрямую).

Процедура выполнения глобальной транзакции, запущенной на сайте S1, будет выглядеть следующим образом.

1. Координатор транзакций (TC1) на сайте S1 разделит транзакции на несколько субтранзакций, исходя из информации, сохраняемой в глобальном каталоге системы.
2. Компонент передачи данных на сайте S1 отправит описание соответствующих субтранзакций на сайты S2 и S3.
3. Координаторы транзакций на сайтах S2 и S3 организуют выполнение поступивших субтранзакций. Результаты их выполнения будут отправлены координатору TC1 с помощью компонентов передачи данных. Весь описанный процесс схематически представлен на рис. 20.1.

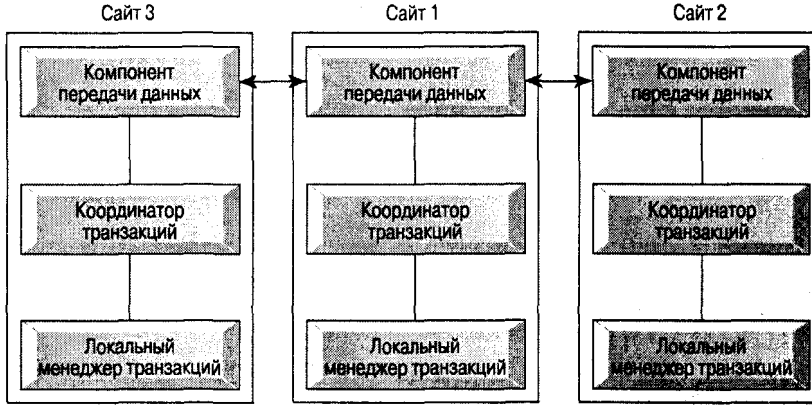


Рис. 20.1. Схема координации выполнения распределенных транзакций

Основываясь на приведенной выше общей схеме обработки распределенных транзакций, мы можем перейти к обсуждению протоколов управления параллельностью, обнаружения взаимных блокировок и восстановления.

20.2. Управление распределенной параллельностью

В этом разделе будут представлены протоколы, которые могут использоваться для организации управления параллельностью в распределенных СУБД. Однако прежде всего следует познакомиться с целями, которые стоят перед механизмом управления распределенной параллельностью.

20.2.1. Цели управления распределенной параллельностью

Если предположить, что в системе не существует отказов, то назначение механизма управления параллельностью будет состоять в предоставлении гарантий сохранения целостности данных и завершения каждого атомарного действия в пределах некоторого установленного промежутка времени. Кроме того, хороший механизм управления параллельностью в распределенной СУБД должен обеспечивать следующее:

- устойчивость к отказам на сайте и в линиях связи;
- высокий уровень параллельности, удовлетворяющий существующим требованиям производительности;
- невысокий дополнительный уровень потребления времени процессора и других системных ресурсов;
- удовлетворительные показатели работы с сетевыми соединениями, имеющими высокое значение времени задержки соединения;
- отсутствие дополнительных ограничений на структуру атомарных действий (Kohler, 1981).

В разделе 17.2.1 мы обсудили те типы проблем, которые могут возникнуть, когда несколько пользователей имеют одновременный доступ к базе данных. К ним относятся проблемы потерянного обновления, зависимости от промежуточных результатов и несогласованности обработки. Все эти проблемы также имеют место и в распределенной среде, однако здесь существуют еще и трудности, вызванные распределенным хранением данных. Одна из них называется **проблемой согласованности многих копий данных** и возникает в тех случаях, когда существует больше одной копии некоторого элемента данных, размещенных в различных местоположениях. Очевидно, что для поддержки согласованности глобальной базы данных при обновлении реплицируемого элемента данных на одном из сайтов необходимо отразить это изменение и во всех остальных копиях данного элемента. Если обновление не будет отражено во всех копиях, база данных перейдет в несогласованное состояние. В данном разделе мы будем предполагать, что обновление реплицируемых элементов выполняется в системе *синхронно*, как часть транзакции, включающей исходную операцию обновления. В разделе 20.6 мы вернемся к этому вопросу и обсудим способы *асинхронного* проведения обновления реплицируемых элементов, т.е. через некоторое время после завершения транзакции, включающей исходную операцию обновления реплицируемого элемента данных.

20.2.2. Распределенная упорядоченность

Концепция упорядоченности, обсуждавшаяся нами в разделе 17.2.2, может быть расширена с целью ее применения и в случае распределенности среды хранения данных. Если график выполнения транзакций на каждом из сайтов является упорядоченным, то **глобальный график** (представляющий собой объединение всех локальных графиков) также будет упорядоченным и предоставляющим идентичную локальную упорядоченность. Для этого необходимо, чтобы все субтранзакции располагались в одном и том же порядке в эквивалентном последовательном графике на всех сайтах. Следовательно, если субтранзакцию T_i на сайте S_1 обозначить как T_i1, то следует гарантировать, что если $T_i1 < T_j1$, то $T_ix < T_jx$ для всех сайтов Sx, на которых транзакции T_i и T_j имеют субтранзакции.

Решения по организации управления параллельностью в распределенной среде строятся на двух основных подходах: использовании механизма блокировок и использовании временных отметок. Оба они уже были рассмотрены нами в разделе 17.2 в отношении централизованных баз данных. Поэтому заданное множество транзакций может выполняться параллельно в одном из следующих случаев.

- Механизм блокировки гарантирует, что график параллельного выполнения транзакций будет эквивалентен некоторому (непредсказуемому) варианту последовательного выполнения этих транзакций.
- Механизм обработки временных меток гарантирует, что график параллельного выполнения транзакций будет эквивалентен *конкретному* варианту последовательного выполнения этих транзакций в соответствии с их временными метками.

Если база данных является либо централизованной, либо фрагментированной, но не использующей репликации данных, то в ней существует только одна копия каждого элемента данных. Поэтому все выполняемые в ней транзакции являются либо локальными, либо могут быть выполнены на одном из удаленных сайтов. В этом случае могут использоваться протоколы, обсуждавшиеся нами в разделе 17.2. Однако эти протоколы должны быть модифицированы, если в системе существует репликация данных или выполняются транзакции, включающие доступ к элементам данных, размещенным более чем на одном сайте. В случае применения протоколов, использующих механизм блокировок, дополнительно следует гарантировать устранение в системе ситуаций взаимных блокировок. Причем потребуется обнаружение взаимных блокировок не только на каждом из локальных уровней, но и на глобальном уровне, когда взаимная блокировка возникает при обращении к данным, размещенным на нескольких сайтах. Подробнее речь об этом пойдет в разделе 20.3.

20.2.3. Блокирующие протоколы

В этом разделе мы обсудим некоторые протоколы двухфазной блокировки (2PL), которые могут применяться для обеспечения упорядоченности графиков в среде распределенных СУБД. К ним относятся централизованный протокол двухфазной блокировки, двухфазная блокировка с первичными копиями, распределенный протокол двухфазной блокировки и блокирование большинства.

Централизованный протокол двухфазной блокировки

При использовании этого протокола существует единственный сайт, на котором сохраняется вся информация о блокировании элементов данных в системе (Alsberg and Day, 1976; Garsia-Molina, 1979). Соответственно во всей распределенной СУБД существует только один планировщик, или *менеджер блокировки*, способный устанавливать и снимать блокировку с элементов данных. При запуске глобальной транзакции на сайте S_1 централизованный протокол двухфазной блокировки работает следующим образом.

1. Координатор транзакций на сайте S_1 разделяет глобальную транзакцию на несколько субтранзакций, используя информацию, сохраняемую в глобальном системном каталоге. Координатор отвечает за соблюдение согласованности базы данных. Если транзакция включает обновление реплицируемого элемента данных, координатор должен гарантировать, что все существующие копии этого элемента данных будут обновлены. Поэтому координатор должен потребовать установить блокировку для записи для всех существующих копий обновляемого элемента данных — только после этого он сможет приступить к обновлению каждой копии и ее освобождению. Для чтения обновляемого элемента данных координатор может выбрать любую из существующих копий. Обычно считывается локальная копия, если таковая существует.
2. Локальные менеджеры транзакций приступают к выполнению запросов этой глобальной транзакции и снимают блокировку элементов данных, обращаясь к центральному менеджеру блокировки с использованием обычных правил протокола двухфазной блокировки.
3. Центральный менеджер блокировки проверяет совместимость поступающих запросов на блокировку элементов данных с текущим состоянием блокировки этих элементов. Если это так, менеджер блокировки направляет на исходный сайт со-

общение, уведомляющее, что требуемая блокировка элемента данных установлена. В противном случае он помещает запрос в очередь, где он и находится вплоть до того момента, когда требуемая блокировка сможет быть предоставлена.

Вариантом этой схемы является случай, когда координатор транзакций выполняет все запросы на блокировку от имени локального менеджера транзакций. В этом случае менеджер запросов взаимодействует только с координатором транзакций, а не с отдельными локальными менеджерами транзакций.

Преимущество централизованного протокола двухфазной блокировки состоит в том, что его относительно просто реализовать. Обнаружение взаимных блокировок может выполняться теми же методами, что и в централизованных СУБД, поскольку вся информация о блокировках элементов находится в распоряжении единственного менеджера блокировок. Основной недостаток этой схемы связан с тем, что любая централизация в распределенной СУБД автоматически становится узким местом системы и резко снижает уровень ее надежности и устойчивости. Поскольку все запросы на блокировки элементов данных направляются на единственный центральный сайт, его быстродействие ограничивает возможности всей системы. Кроме того, отказ этого сайта вызовет паралич работы всей распределенной системы. Однако этой схеме свойствен относительно невысокий уровень затрат на передачу данных. Например, глобальная операция обновления, создающая агентов (субтранзакции) на n сайтах, при наличии центрального менеджера блокировок может потребовать отправки ему не менее 2n + 3 сообщений:

- один запрос на блокировку;
- одно сообщение о предоставлении блокировки;
- n сообщений с требованием обновления;
- n подтверждений о выполненном обновлении;
- один запрос на снятие блокировки.

Двухфазная блокировка с первичными копиями

В этом варианте протокола попытка преодолеть свойственные централизованному двухфазному протоколу недостатки предпринимается за счет распределения единственного менеджера блокировки по нескольким сайтам. В данном случае каждый локальный менеджер отвечает за управление блокировкой некоторого набора элементов данных. Для каждого реплицируемого набора данных одна из копий выбирается в качестве **первичной копии** (primary copy), а все остальные расцениваются как **ведомые** (slave copy). Выбор первичного сайта может осуществляться по разным правилам, причем сайт, который выбран для управления блокировкой первичной копии данных, необязательно должен содержать саму эту копию (Stonebraker and Neuhold, 1977).

Данный протокол является простым расширением централизованного протокола двухфазной блокировки. Основное различие состоит в том, что когда элемент данных обновляется, координатор транзакций должен определить, где находится его первичная копия, и послать запрос на блокировку элемента соответствующему менеджеру блокировки. При обновлении элемента данных достаточно установить блокировку для записи только его первичной копии. После того как первичная копия будет обновлена, внесенные изменения могут быть распространены на все ведомые копии. Это распространение должно быть выполнено с максимально возможной скоростью, чтобы предотвратить чтение другими транзакциями устаревших значений данных. Однако нет необходимости выполнять все обновления как одну неделимую транзакцию. Данный протокол гарантирует актуальность значений только для первичной копии данных.

Подобный подход может использоваться в тех случаях, когда данные подвергаются репликации избирательно, обновление их происходит относительно нечасто, а сайты не нуждаются в использовании непременно текущей копии всех элементов данных. Недостатками данного подхода являются усложнение методов обнаружения взаимных блокировок в связи с наличием нескольких менеджеров блокировки, а также сохране-

ние в системе определенной степени централизации, поскольку запросы на блокировку первичной копии элемента могут быть выполнены только на единственном сайте. Последний недостаток может быть частично компенсирован за счет объявления резервных сайтов, содержащих копию информации о блокировании элементов. Данный протокол характеризуется меньшими затратами на передачу сообщений и более высоким уровнем производительности, чем централизованный протокол двухфазной блокировки, — в основном за счет снижения уровня удаленной блокировки.

Распределенный протокол двухфазной блокировки

В этом протоколе также предпринимается попытка преодолеть недостатки, свойственные централизованному протоколу двухфазной блокировки, но уже за счет размещения менеджеров блокировки на каждом сайте системы. В данном случае каждый менеджер блокировки отвечает за управление элементами данных, размещенными на его сайте. Если данные не подвергаются репликации, этот протокол функционирует аналогично протоколу двухфазной блокировки с первичными копиями. В противном случае распределенный протокол двухфазной блокировки использует особый протокол управления репликацией, получивший название "один раз считать, обновить всех" (Read-One-Write-All — ROWA). В этом случае для операций считывания может использоваться любая копия реплицируемого элемента, однако, прежде чем значение элемента можно будет обновить, должны быть заблокированы для записи все его существующие копии. В этой схеме управление блокировками осуществляется децентрализованным способом, что позволят освободиться от недостатков, свойственных централизованному управлению. Однако данному подходу свойственны иные недостатки, связанные с существенным усложнением методов выявления взаимных блокировок (из-за наличия многих менеджеров блокировки) и возрастанием издержек на пересылку данных, вызванных необходимостью блокировать все копии каждого обновляемого элемента. В данном протоколе выполнение глобальной операции обновления, имеющей агентов на n сайтах, потребует пересылки не менее 5n сообщений:

- n сообщений с запросами на блокировку;
- n сообщений с предоставлением блокировки;
- n сообщений с требованием обновления элемента;
- n сообщений с подтверждением выполненного обновления;
- n сообщений с запросами на снятие блокировки.

Это количество сообщений может быть сокращено до 4n, если опустить запросы на снятие блокировки, которое в этом случае будет выполняться при обработке операции окончательной фиксации распределенной транзакции. Распределенный протокол двухфазной блокировки был использован в системе System R* (Mohan et al., 1986).

Блокирование большинства

Этот протокол можно считать расширением распределенного протокола двухфазной блокировки, в котором устраняется необходимость блокирования всех копий реплицируемого элемента данных перед его обновлением. В этом случае менеджер блокировки также имеется на каждом из сайтов системы, где он осуществляет управление блокировкой всех данных, размещаемых на этом сайте. Когда транзакции требуется считать или записать элемент данных, реплики которого имеются на n сайтах системы, она должна отправить запрос на блокирование этого элемента более чем на половину всех тех n сайтов, где имеются его копии. Транзакция не имеет права продолжать свое выполнение, пока она не установит блокировку большинства копий элемента данных. Если ей не удастся это сделать за некоторый установленный промежуток времени, она отменяет свои запросы и информирует все сайты об отмене ее выполнения. Если большинство будет получено, все сайты информируются о том,

что требуемый уровень блокировки достигнут. Поскольку блокировка для чтения является разделяемой, любое количество транзакций может одновременно установить блокировку этого типа для большей части копий одного и того же элемента данных. Однако только одна из транзакций сможет установить необходимое количество блокировок копий элемента для записи (Thomas, 1979).

В этом случае также устраняются все недостатки, свойственные централизованному подходу. Однако данному протоколу свойственны собственные недостатки, заключающиеся в повышенной сложности алгоритма, усложнении процедур выявления взаимной блокировки, а также необходимости отправки не менее [(n + 1)/2] сообщений с запросами на установление блокировки и [(n + 1)/2] сообщений с запросами на отмену блокировки. Метод успешно работает, однако он показывает себя излишне жестким в отношении блокировок для чтения. Для выполнения чтения достаточно установить блокировку лишь одной копии элемента данных, а именно той, которая будет прочитана. Однако особенностью данного метода является обязательное требование выполнить блокировку большей части имеющихся копий элемента данных.

20.2.4. Протоколы с временными отметками

Мы уже обсуждали протоколы для централизованных баз данных, использующие временные отметки, в разделе 17.2.5. Задачей подобных протоколов является глобальное упорядочивание транзакций, выполняемое таким образом, что более старые транзакции (имеющие *меньшую* временную отметку) в случае конфликта получают приоритет. В распределенной среде также необходимо будет генерировать уникальные значения временных отметок, причем как локально, так и глобально. Очевидно, что использование на каждом сайте системных часов или накапливаемого счетчика событий (как предлагалось в разделе 17.2.5) уже не является подходящим решением. Часы на каждом сайте могут быть недостаточно синхронизированы, а при использовании счетчиков событий ничто не мешает генерации одних и тех же значений счетчика одновременно на разных сайтах.

Общим подходом в распределенных СУБД является конкатенация локальной временной отметки с уникальным идентификатором сайта по схеме *<локальная_отметка, идентификатор_сайта>*. Значение идентификатора сайта будет иметь меньший вес, что гарантирует упорядочивание событий в соответствии с моментом их возникновения и лишь затем в соответствии с местом их появления. Чтобы предотвратить генерацию более загруженными сайтами больших значений временных меток в сравнении с недогруженными сайтами, необходимо использовать определенный механизм синхронизации значений временных меток сайтов. Каждый сайт помещает свою текущую временную метку в сообщения, посылаемые на другие сайты. При получении сообщения сайт-получатель сравнивает текущее значение его временной метки с полученным и, если его текущая временная метка оказывается меньше, меняет ее значение на некоторое другое, превосходящее то значение временной метки, которое было получено им в сообщении.

20.3. Распределенная взаимная блокировка

Любые алгоритмы управления параллельностью, использующие механизм блокировки (и некоторые алгоритмы с использованием временных отметок, помещающие транзакции в состояние ожидания), могут приводить к появлению в системе ситуации взаимной блокировки процессов (см. раздел 17.2.4). В распределенной среде выявление взаимной блокировки существенно усложняется, если управление блокировкой объектов не является централизованным, что демонстрируется в приведенном ниже примере.

Пример 20.1. Взаимная блокировка в распределенной среде

Рассмотрим три транзакции, T_1, T_2 и T_3, имеющие следующие характеристики:

- транзакция T_1 запускается на сайте S_1 и создает агента на сайте S_2;
- транзакция T_2 запускается на сайте S_2 и создает агента на сайте S_3;
- транзакция T_3 запускается на сайте S_3 и создает агента на сайте S_1.

Эти транзакции устанавливают блокировки для чтения и записи по приведенной ниже схеме, где обозначение read_lock(T_i, x_j) означает установку транзакцией T_i блокировки элемента данных x_j для чтения, а обозначение write_lock(T_i, x_j) означает установку транзакцией T_i блокировки элемента данных x_j для записи.

Время	Сайт S_1	Сайт S_2	Сайт S_3
t_1	read_lock(T_1, x_1)	write_lock(T_2, y_2)	read_lock(T_3, z_3)
t_2	write_lock(T_1, y_1)	write_lock(T_2, z_2)	
t_3	write_lock(T_3, x_1)	write_lock(T_1, y_2)	write_lock(T_2, z_3)

Для каждого из сайтов мы можем построить граф ожидания (как показано на рис. 20.2).

Ни один из этих локальных графов не содержит циклов, что может быть расценено как свидетельство отсутствия в системе процессов взаимных блокировок. Однако это не так, поскольку после объединения индивидуальных графов в единый глобальный граф ожидания, представленный на рис. 20.3, в нем обнаруживается петля, указывающая на наличие взаимной блокировки по следующей схеме:

$T_3 \rightarrow T_1 \rightarrow T_2 \rightarrow T_3$

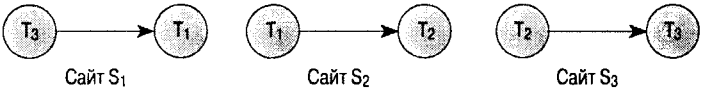


Рис. 20.2. Графы ожидания для сайтов S_1, S_2 и S_3

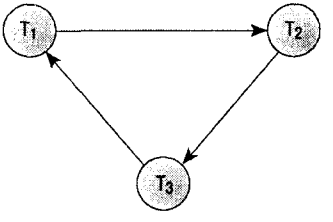


Рис. 20.3. Комбинированный граф ожидания для сайтов S_1, S_2 и S_3

Приведенный выше пример показывает, что в распределенной СУБД для выявления ситуаций взаимной блокировки недостаточно использовать обычные графы ожидания, построенные на каждом из сайтов локально. Необходимо также строить глобальный граф ожидания, представляющий собой объединение всех локальных графов ожидания. Существуют три общих метода выявления взаимных блокировок в распределенных СУБД: **централизованный, иерархический и распределенный.**

Централизованный метод выявления взаимных блокировок

При централизованном выявлении взаимных блокировок один из сайтов системы назначается координатором выявления взаимных блокировок (Deadlock Detection Coordinator, DDC). Сайт DDC отвечает за построение и обработку глобального графа ожидания. С определенным интервалом каждый менеджер блокировки в системе направляет в адрес DDC свой локальный граф ожидания. Сайт DDC выполняет построение глобального графа ожидания и проверяет его на наличие циклов. Если граф ожидания содержит один или более циклов, DDC должен разрушить их, выбрав те транзакции, которые должны быть отменены с выполнением отката, а затем перезапущены. В обязанности DDC входит информирование всех сайтов, принимающих участие в обработке отменяемых транзакций, о том, что последние необходимо отменить и перезапустить.

Для минимизации количества пересылаемых данных каждый менеджер блокировки посылает в адрес DDC только сведения об изменениях в локальном графе ожидания, произошедших с момента предыдущей отправки сведений. Посылаемые сведения включают информацию только о добавлении или удалении ребер в локальном графе ожидания. Недостатком централизованного подхода является то, что он снижает надежность всей системы, поскольку отказ центрального сайта может вызвать большие проблемы в функционировании всей системы.

Иерархический метод выявления взаимных блокировок

При иерархическом методе выявления взаимных блокировок в системе сайты в сети образуют некоторую иерархию. Каждый из сайтов для выявления наличия взаимных блокировок посылает свой локальный граф ожидания на сайт, расположенный в иерархии на уровень выше его (Menasce and Muntz, 1979). На рис. 20.4 представлена иерархия из восьми сайтов, от S_1 до S_8. Первый уровень иерархии образуют все восемь сайтов, каждый из которых выполняет локальный контроль наличия взаимных блокировок. Второй уровень образуют узлы DD_{ij}, выполняющие обнаружение взаимных блокировок для соседней пары сайтов S_i и S_j. Третий уровень образуют узлы, выполняющие контроль взаимных блокировок на четырех соседних узлах. Корнем дерева является глобальный детектор взаимных блокировок, способный обнаружить взаимную блокировку между любыми сайтами системы, например между сайтами S_1 и S_8.

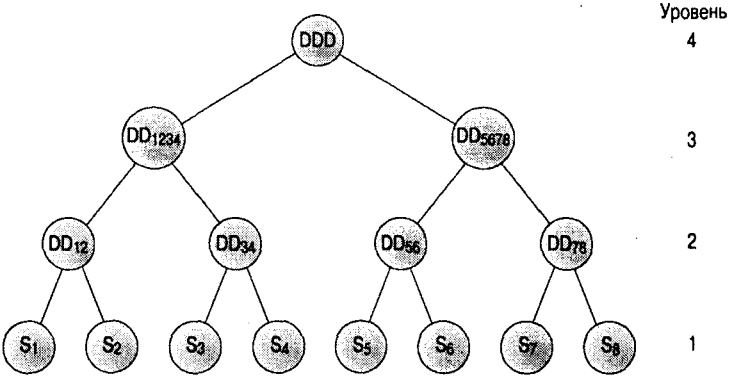


Рис. 20.4. Схема построения иерархического механизма выявления взаимных блокировок

Иерархический подход сокращает зависимость выявления взаимных блокировок от центрального сайта, а также способствует снижению издержек на передачу данных. Однако его реализация намного сложнее, особенно с учетом возможности отказов отдельных сайтов и линий связи.

Распределенный метод выявления взаимных блокировок

Существуют различные варианты алгоритмов распределенного выявления взаимных блокировок, однако здесь мы рассмотрим только наиболее популярный из них — метод выявления взаимных блокировок, разработанный Обермарком (Obermarck, 1982). В этом методе к локальному графу ожидания добавляется внешний узел T_{ext}, отражающий наличие агента на удаленном сайте. Когда транзакция T_1 на сайте S_1 создает агента на другом сайте, например S_2, то к локальному графу ожидания добавляется ребро, соединяющее узел T_1 с узлом T_{ext}. Аналогичным образом на сайте S_2 к локальному графу добавляется ребро, соединяющее узел T_{ext} с узлом T_1.

Например, глобальный граф ожидания, показанный на рис. 20.3, может быть представлен в виде локальных графов ожидания на сайтах S_1, S_2 и S_3, как показано на рис. 20.5. Ребра, соединяющие узлы локального графа с узлом T_{ext}, помечены с указанием имен соответствующего сайта. Например, ребро, соединяющее узлы T_1 и T_{ext} на сайте S_1, помечается меткой S_2, поскольку данное ребро представляет агента, созданного транзакцией T_1 на сайте S_2.

Если локальный граф ожидания содержит цикл, не включающий узел T_{ext}, то на сайте существует локальная ситуация взаимной блокировки. Если цикл на локальном графе ожидания включает узел T_{ext}, то потенциально в системе может присутствовать ситуация глобальной взаимной блокировки. Однако существование подобного цикла не обязательно означает наличие глобальной взаимной блокировки, поскольку узел T_{ext} может представлять несколько различных агентов. Тем не менее при действительном наличии глобальной взаимной блокировки подобная петля в локальном графе присутствует обязательно. Для уточнения, действительно ли имеет место глобальная взаимная блокировка, локальные графы должны быть слиты. Если для сайта S_i потенциально возможна глобальная взаимная блокировка, то его локальный граф ожидания будет иметь следующий вид:

$$T_{ext} \rightarrow T_i \rightarrow T_j \rightarrow \ldots \rightarrow T_k \rightarrow T_{ext}$$

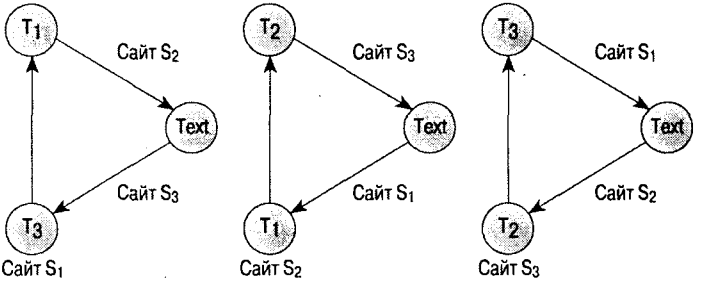


Рис. 20.5. Распределенный метод выявления глобальной взаимной блокировки

Для того чтобы предотвратить взаимную пересылку сайтами их графов ожидания, используется простой алгоритм. По этому алгоритму каждой из транзакций присваивается временная отметка и устанавливается правило, согласно которому сайт S_i пересылает свой граф ожидания только тому сайту, скажем S_k, на котором ожидает транзакция T_k. Другими словами, если $t_s(T_i) < t_s(T_k)$. Если обнаруживается, что $t_s(T_i) < t_s(T_k)$, то для проверки наличия взаимной блокировки сайт S_i должен переслать свой локальный граф ожидания на сайт S_k. Сайт S_k добавляет полученную информацию к своему локальному графу ожидания и проверяет результат на наличие циклов, не включающих узел T_{ext}. Если подобных циклов не обнаружено, процесс продолжается либо до появления цикла (и в этом случае одна или более транзакций

откатываются и перезапускаются с повторным созданием агентов), либо до построения полного глобального графа ожидания, не содержащего циклов. В последнем случае взаимные блокировки в системе отсутствуют. Обермарк доказал, что если в системе существует глобальная взаимная блокировка, то описанная выше процедура непременно вызовет появление цикла в графе ожидания одного из сайтов.

Три локальных графа ожидания, показанные на рис. 20.5, содержат следующие циклы:

S_1: $T_{ext} \rightarrow T_3 \rightarrow T_1 \rightarrow T_{ext}$
S_2: $T_{ext} \rightarrow T_1 \rightarrow T_2 \rightarrow T_{ext}$
S_3: $T_{ext} \rightarrow T_2 \rightarrow T_3 \rightarrow T_{ext}$

В данном примере локальный граф ожидания сайта S_1 может быть отослан на тот сайт, на котором ожидает транзакция T_1, т.е. на сайт S_2. Локальный граф ожидания сайта S_2 пополняется полученной информацией и приобретает следующий вид:

S_2: $T_{ext} \rightarrow T_3 \rightarrow T_1 \rightarrow T_2 \rightarrow T_{ext}$

Этот граф также указывает на возможность существования в системе взаимной блокировки, поэтому сайт S_2 отсылает свой граф ожидания на тот сайт, на котором ожидает транзакция T_2, а именно на сайт S_3. Локальный граф ожидания сайта S_3 пополняется поступившей информацией и приобретает такой вид:

S_3: $T_{ext} \rightarrow T_3 \rightarrow T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_{ext}$

Этот глобальный граф ожидания содержит цикл, не включающий узел T_{ext}. В результате можно сделать вывод о наличии в системе ситуации взаимной блокировки и предпринять необходимые меры для ее устранения. Распределенный метод выявления взаимной блокировки потенциально более устойчив, чем иерархический и централизованный методы, однако поскольку ни один из сайтов не содержит всей информации, необходимой для выявления ситуации взаимной блокировки, это может явиться причиной повышенного уровня обмена информацией между сайтами.

20.4. Восстановление распределенных баз данных

В этом разделе мы обсудим протоколы, которые используются для обработки отказов, возникающих в распределенной вычислительной среде.

20.4.1. Отказы в распределенной среде

В разделе 19.5.2 были названы четыре типа отказов, которые характерны для распределенных СУБД:

- потеря сообщения;
- отказ линии связи;
- аварийный останов одного из сайтов;
- расчленение сети.

Потеря сообщений или неверная упорядоченность их поступления контролируется используемым базовым сетевым протоколом. Поэтому мы можем сделать законное предположение о прозрачности обработки этих ошибок для компонента передачи данных СУРБД. Ниже мы сосредоточим внимание только на оставшихся трех типах отказов.

Функционирование СУРБД в значительной мере зависит от надежного взаимодействия всех сайтов в сети. В недалеком прошлом линии связи часто оказывались недостаточно надежными. Хотя с тех пор сетевые технологии были существенно улучшены и надежность сетей значительно возросла, тем не менее, отказы линий связи по-прежнему имеют место. В частности, отказ линии связи может привести к расчленению сети на от-

дельные фрагменты, в результате чего сайты одного фрагмента смогут взаимодействовать, но не будут иметь доступа к сайтам в другом фрагменте. Пример подобной ситуации показан на рис. 20.6. Здесь исходная распределенная система из пяти сайтов, показанная на рис. 20.6, а, в результате отказа линии связи между сайтами 1 и 2 была расчленена на два фрагмента: сайты 1, 4, 5 и сайты 2, 3 — как показано на рис. 20.6, б.

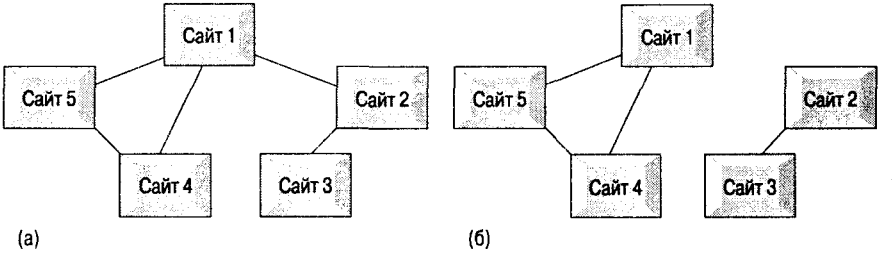


Рис. 20.6. Расчленение сети: а) исходное состояние сети до отказа; б) состояние сети после отказа линии связи

В некоторых случаях бывает трудно установить, что именно отказало — линия связи или сайт, с которым она соединена. Например, предположим, что сайт S_1 не может взаимодействовать с сайтом S_2 в пределах установленного временного интервала (тайм-аута). Причиной этому может служить любая из следующих ситуаций:

- сайт S_2 не функционирует или нарушена работа сети;
- отказ в линии связи;
- имеет место расчленение сети;
- сайт S_2 в настоящее время перегружен и не способен ответить на поступившее сообщение в пределах установленного тайм-аута.

Выбор корректного значения тайм-аута, которое с уверенностью позволит утверждать о действительной невозможности взаимодействия двух сайтов, является непростой задачей.

20.4.2. Влияние отказов на процедуры восстановления

Как и в случае локального управления восстановлением, распределенные средства восстановления предназначены для обеспечения свойств **атомарности** и **длительности** распределенных транзакций. Для достижения атомарности глобальных транзакций СУРБД должна гарантировать, что субтранзакции глобальной транзакции будут либо все зафиксированы, либо все отменены. Если СУРБД обнаружит, что некоторый сайт отказал или стал недоступен, она должна будет выполнить следующие действия.

- Отменить выполнение всех транзакций, затронутых данным отказом.
- Отметить сайт как отказавший, чтобы предотвратить любые попытки его использования другими сайтами.
- Периодически проверять состояние отказавшего сайта для восстановления его функционирования или же, наоборот, ожидать поступления от этого сайта широковещательного сообщения с указанием о восстановлении его нормальной работы.
- При перезапуске сайта после отказа на нем должна выполняться процедура восстановления, предназначенная для отката любых транзакций, выполненных на момент отказа лишь частично.
- После завершения процедуры локального восстановления отказавший сайт должен обновить свою копию базы данных, чтобы привести ее в соответствие с остальной частью системы.

Если имеет место расчленение сети, как в приведенном выше примере, СУРБД должна гарантировать, что если агенты некоторой глобальной транзакции оказались активными в разных фрагментах, то сайт S_1 и другие сайты одного фрагмента должны быть лишены возможности зафиксировать результаты этой глобальной транзакции, поскольку сайт S_2 и прочие сайты другого фрагмента могут принять решение выполнить ее откат. В результате атомарность глобальной транзакции будет нарушена.

Распределенные протоколы восстановления

Как уже упоминалось выше, процессы восстановления в распределенных СУБД усложняются тем фактом, что соблюдение свойства атомарности требуется как в отношении локальных субтранзакций, так и всей глобальной транзакции в целом. Механизмы восстановления, описанные в разделе 17.3, гарантируют атомарность лишь субтранзакций, тогда как СУРБД необходимо гарантировать и атомарность всей глобальной транзакции. По этой причине в процедуры фиксации и отката транзакций необходимо внести такие изменения, которые не позволят глобальной транзакции зафиксировать или отменить результаты ее выполнения, пока все ее субтранзакции не будут успешно зафиксированы или отменены. Кроме того, модифицированные протоколы должны обрабатывать ситуации отказа сайта или линии связи таким образом, чтобы гарантировать, что отказ одного из сайтов не окажет влияния на обработку данных на другом сайте. Другими словами, работоспособные сайты не должны оказаться заблокированными из-за отказа других сайтов. Протоколы, обеспечивающие выполнение последнего требования, называются **неблокирующими**. В этом разделе мы рассмотрим два широко распространенных протокола фиксации транзакций, которые могут использоваться в среде СУРБД: протокол двухфазной фиксации транзакций (2PC) и неблокирующий протокол трехфазной фиксации транзакций (3PC).

Мы будем предполагать, что каждая глобальная транзакция связана с некоторым сайтом, функционирующим как **координатор** выполнения этой транзакции. Обычно координатором является тот сайт, на котором транзакция была инициирована. Сайты, на которых глобальная транзакция создала агентов, мы будем называть **участниками**. Предполагается, что координатор транзакции знает идентификаторы всех участников, а каждый участник знает идентификатор координатора, но не обязан знать идентификаторы остальных участников.

20.4.3. Двухфазная фиксация транзакций (2PC)

Как следует из названия данного протокола, фиксация результатов транзакций выполняется в два этапа: **этап голосования** (voting phase) и **принятия решения** (decision phase). Основная идея состоит в том, что координатор должен опросить всех участников, готовы ли они к фиксации транзакции. Если хотя бы один из участников потребует отката или не ответит на запрос в пределах установленного тайм-аута, координатор укажет всем участникам на необходимость выполнить откат данной транзакции. Глобальное решение должно быть принято *всеми* участниками. Если некоторый участник требует отката транзакции, то он имеет право выполнить его немедленно. Фактически любой сайт имеет право откатить свою субтранзакцию в любое время, вплоть до того момента, пока он не пошлет согласие на ее фиксацию. Подобный тип отката называют **односторонним откатом**. Если участник проголосовал за фиксацию транзакции, то он должен ожидать до тех пор, пока координатор не пошлет широковещательное сообщение либо о глобальной фиксации, либо о глобальном откате этой транзакции. Данный протокол предполагает, что каждый сайт имеет свой собственный локальный журнал и с его помощью может надежно откатить или зафиксировать транзакцию. Двухфазный протокол фиксации транзакций включает этап ожидания сообщения от других сайтов. Во избежание нежелательных блокировок процессов система использует механизм обнаружения тайм-аута. Для фиксации глобальной транзакции координатор выполняет следующую процедуру.

Фаза 1

1. Занести запись begin_commit в системный журнал и обеспечить ее перенос из буфера во вторичную память. Послать всем участникам команду PREPARE. Ожидать ответов всех участников в пределах установленного тайм-аута.

Фаза 2

2. Если участник возвращает сообщение ABORT, поместить в системный журнал запись abort и обеспечить ее перенос из буфера во вторичную память. Послать всем участникам команду GLOBAL_ABORT. Ожидать ответов всех участников в пределах установленного тайм-аута.
3. Если участник возвращает сообщение READY_COMMIT, обновить список участников, приславших свои ответы. Если ответы прислали все участники, поместить в системный журнал запись commit и обеспечить ее перенос из буфера во вторичную память. Послать всем участникам команду GLOBAL_COMMIT. Ожидать ответов всех участников в пределах установленного тайм-аута.
4. После поступления всех подтверждений о фиксации транзакции поместить в системный журнал запись end_transaction. Если некоторые сайты не прислали подтверждения о фиксации, заново направить на эти сайты сообщение о принятом глобальном решении и поступать по этой схеме до получения всех требуемых подтверждений.

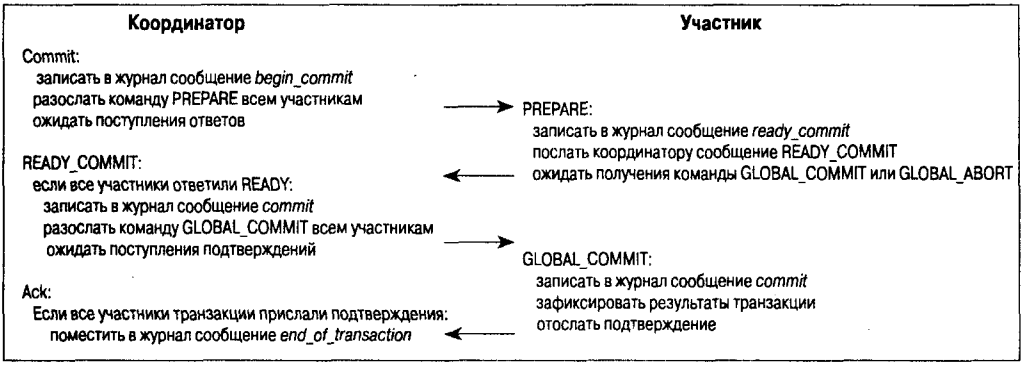
Координатор должен ждать поступления результатов голосования от всех участников. Если сайт не прислал сообщения с его решением, координатор по умолчанию предполагает поступление от этого сайта решения об откате транзакции и рассылает всем участникам широковещательное сообщение GLOBAL_ABORT. Вопрос о действиях, которые должен предпринять участник при рестарте после отказа, мы рассмотрим чуть ниже. При фиксации транзакции участник выполняет следующую процедуру.

1. При получении участником команды PREPARE он выполняет одно из следующих действий:
   а) помещает запись ready_commit в файл журнала и переносит из буфера во вторичную память все записи, относящиеся к данной транзакции. Отправляет координатору сообщение READY_COMMIT;
   б) помещает запись abort в файл журнала и переносит ее из буфера во вторичную память. Отправляет координатору сообщение ABORT. Выполняет односторонний откат транзакции.

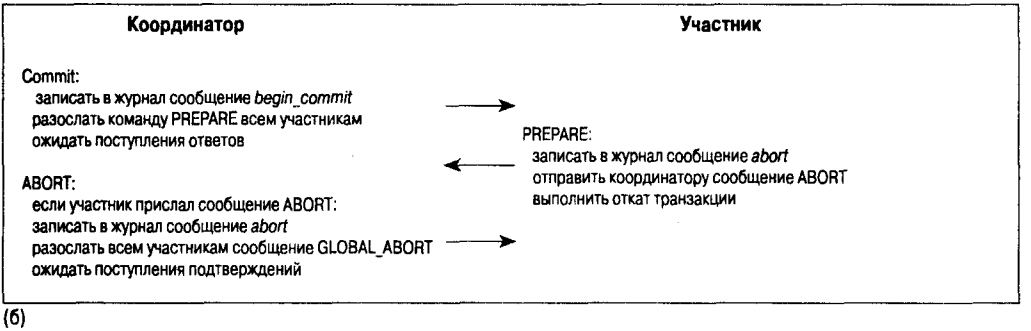
   Ожидает ответа координатора в пределах установленного тайм-аута.
2. Если участник получает команду GLOBAL_ABORT, то он помещает запись abort в файл журнала и переносит ее из буфера во вторичную память. Затем выполняется откат транзакции и по его завершении координатору посылается соответствующее подтверждение.
3. Если участник получает команду GLOBAL_COMMIT, то он помещает запись commit в файл журнала и переносит ее из буфера во вторичную память. Затем выполняется фиксация транзакции, освобождение всех установленных и удерживаемых блокировок, после чего координатору посылается соответствующее подтверждение.

Если участнику не удалось получить от координатора команду с результатами проведения голосования, то по истечении установленного тайм-аута он просто откатывает данную транзакцию. Следовательно, еще до поступления команды с результатами голосования участник может принять решение об откате субтранзакции и выполнить его. Все описанные процедуры взаимодействия координатора и участника представлены на рис. 20.7.

Участник должен ожидать поступления от координатора команды GLOBAL_COMMIT или GLOBAL_ ABORT. Если в пределах установленного тайм-аута он не получит никакой команды или координатор не получит ответа от участника, то предполагается, что на соответствующем сайте произошел отказ и в работу запускается **протокол ликвидации** (termination protocol). Только функционирующие сайты выполняют протокол ликвидации, на отказавших сайтах после рестарта выполняется **протокол восстановления.**



(а)



(б)

Рис. 20.7. Двухфазная фиксация транзакций: а) когда все участники проголосовали за фиксацию транзакции; б) когда некоторый участник голосует за откат транзакции

Протоколы ликвидации

Протокол ликвидации запускается всякий раз, когда координатор или участник не получает ожидаемого сообщения до наступления тайм-аута. Предпринимаемые действия зависят от того, кто не получил сообщения, координатор или участник, и какое именно сообщение не было получено.

Координатор

В процессе выполнения фиксации транзакции координатор может находиться в одном из четырех состояний: INITOAL, WAITONG, DECIDED и COMPLITED — как показано на диаграмме состояния, представленной на рис. 20.8, а. Однако состояние тайм-аута может иметь место только в двух случаях из четырех. В подобных случаях предпринимаются следующие действия.

- *Тайм-аут в состоянии WAITING.* Координатор ожидает поступления от всех участников уведомлений о решении, которое они приняли в отношении фиксации или отката данной транзакции. В подобной ситуации координатор не может зафиксировать транзакцию, поскольку он не получил всех

требуемых подтверждений, поэтому он организует действия по глобальному откату данной транзакции.

- *Тайм-аут в состоянии* DECIDED. Координатор ожидает поступления от всех участников уведомлений об успешном откате или фиксации данной транзакции. В подобной ситуации координатор просто повторно рассылает сведения о принятом глобальном решении на все сайты, не приславшие ожидаемого подтверждения.

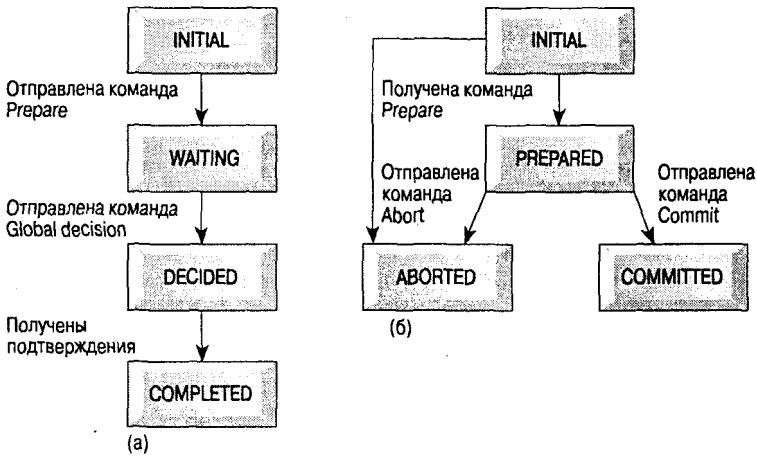


Рис. 20.8. Диаграммы состояния транзакций при использовании протокола двухфазной фиксации: а) координатор; б) участник

Участник

Простейший протокол ликвидации для участника состоит в сохранении процесса на стороне участника заблокированным до тех пор, пока соединение с координатором не будет восстановлено. После этого участник сможет получить информацию о принятом глобальном решении и возобновить обработку транзакции соответствующим образом. Однако из соображений повышения производительности системы на стороне участника могут быть предприняты и другие действия.

В процессе выполнения фиксации транзакции участник может находиться в одном из четырех состояний: INITIAL, PREPARED, ABORTED и COMMITTED — как показано на диаграмме состояния, представленной на рис. 20.8, б. Однако состояние тайм-аута может иметь место только в первых двух случаях из четырех.

- *Тайм-аут в состоянии INITIAL.* Участник ожидает от координатора поступления команды PREPARE. Ее отсутствие может свидетельствовать о том, что сайт координатора отказал, когда процесс фиксации транзакции находился в состоянии INITIAL. В этом случае участник может выполнить односторонний откат транзакции. При поступлении впоследствии команды PREPARE он сможет либо игнорировать ее (в результате чего координатор организует откат транзакции вследствие тайм-аута), либо отправить координатору сообщение ABORT.

- *Тайм-аут в состоянии PREPARED.* Участник ожидает поступления от координатора указаний о глобальной фиксации или глобальном откате данной транзакции. Участник уже известил координатора о своем решении зафиксировать транзакцию, поэтому не имеет права изменить свое мнение и откатить ее. Но он также не может пойти дальше и действительно зафиксировать транзакцию, поскольку глобальным решением может оказаться требование ее отката. До получения дополнительной информации участник оказывается заблокированным. Однако участник может обратиться к каж-

дому из остальных участников транзакции, чтобы получить от одного из них сведения о принятом глобальном решении. Этот вариант известен как **кооперативный протокол ликвидации**. Самый простой способ сообщить участникам о том, кто еще участвует в выполнении транзакции, состоит в дополнении команды PREPARE списком сайтов-участников.

Хотя кооперативный протокол ликвидации снижает вероятность возникновения блокировки, эта ситуация по-прежнему остается возможной и будет сохраняться вплоть до устранения отказа. Если отказ произошел только на сайте координатора (все остальные участники могут установить это, выполняя протокол ликвидации), то они смогут выбрать нового координатора и закончить обработку транзакции описанным выше способом.

Протоколы восстановления

Обсудив действия, которые должны быть предприняты на функционирующем сайте в случае обнаружения отказа другого сайта, мы переходим к рассмотрению действий, которые необходимо выполнить при восстановлении работы сайта после его отказа. Действия, которые выполняются при перезапуске системы, опять-таки зависят от того, какие функции выполнял данный сайт в момент отказа, — координатора или участника транзакции.

Отказ координатора

Мы рассмотрим три различных варианта момента отказа сайта-координатора.

1. *Отказ в состоянии INITIAL.* Координатор еще не начал процедуру фиксации транзакции. В данном случае восстановление заключается в запуске этой процедуры фиксации.
2. *Отказ в состоянии WAITING.* Координатор уже направил команды PREPARE сайтам-участникам и хотя получил еще не все ответы, ни одного предложения об откате получено не было. В этом случае восстановление заключается в повторном запуске процедуры фиксации транзакции.
3. *Отказ в состоянии DECIDED.* Координатор уже направил участникам транзакции указания о ее глобальной фиксации или глобальном откате. Если после рестарта координатор получит все необходимые подтверждения, завершение транзакции можно считать успешным. В противном случае потребуется прибегнуть к протоколу ликвидации, уже обсуждавшемуся выше.

Отказ участника

Целью применения протокола восстановления на сайте-участнике является получение гарантий, что после рестарта системы участник выполнит в отношении транзакции те же самые действия, что и все остальные участники ее выполнения, и эти действия могут быть выполнены независимо (т.е. без необходимости проведения консультаций с координатором или другими участниками). Мы рассмотрим три различных варианта момента отказа сайта-участника.

1. *Отказ в состоянии INITIAL.* Участник еще не успел проголосовать по поводу типа завершения транзакции. Следовательно, в процессе восстановления он может выполнить односторонний откат транзакции, поскольку без голоса данного сайта координатор не мог принять решение о глобальной фиксации этой транзакции.
2. *Отказ в состоянии PREPARED.* Участник уже направил сведения о своем решении в адрес координатора. В этом случае восстановление заключается в применении протокола ликвидации, обсуждавшегося выше.
3. *Отказ в состоянии ABORTED/COMMITTED.* Участник уже завершил обработку транзакции. Следовательно, после рестарта никаких дополнительных действий не потребуется.

Протоколы выбора

Если участники обнаруживают отказ координатора (посредством тайм-аута), они могут выбрать другой сайт, который должен будет взять на себя роль координатора. Один из протоколов выбора требует, чтобы сайты системы были последовательно упорядочены. Предположим, что сайт S_i занимает позицию i в общей последовательности, первым в которой является координатор, а все остальные сайты знают идентификаторы и порядковые номера других сайтов системы, некоторые из которых также могут находиться в состоянии отказа. Данный протокол выбора требует, чтобы каждый функционирующий участник посылал сообщения на сайты с большим идентификационным номером. Следовательно, сайт S_i должен отправить сообщения на сайты S_{i+1}, S_{i+2}, ... , S_n, причем именно в этом порядке. Если сайт S_k получит сообщение от сайта-участника с меньшим номером, то сайт S_k приходит к заключению, что он не может быть новым координатором, и прекращает отправку сообщений.

Данный протокол является относительно эффективным, и большинство участников очень быстро прекращают отправку сообщений. Рано или поздно каждый из участников узнает, существует ли в системе функционирующий сайт-участник с меньшим номером. Ели его нет, данный сайт принимает на себя функции нового координатора транзакции. Если вновь избранный координатор также попадет в состояние тайм-аута, протокол выбора будет запущен еще раз.

После восстановления отказавший сайт немедленно запускает протокол выбора. Если в системе не окажется функционирующего сайта с меньшим номером, данный сайт вынуждает все сайты с большим номером позволить ему стать новым координатором транзакции, независимо от того, существовал уже новый координатор или нет.

Топология взаимодействия сайтов при двухфазной фиксации транзакций

Существует несколько различных способов обмена сообщениями или топологий взаимодействия, которые могут использоваться при реализации двухфазной фиксации транзакций. Один вариант, обсуждавшийся выше, носит название **централизованной двухфазной фиксации**, поскольку все взаимодействия осуществляются через сайт-координатор по схеме, показанной на рис. 20.9, а. Было предложено несколько улучшений централизованного протокола двухфазной фиксации транзакций, позволяющих повысить общую производительность системы либо за счет сокращения количества рассылаемых сообщений, либо за счет ускорения процесса принятия решения. Все эти улучшения реализуются посредством применения различных способов обмена сообщениями.

Одним из альтернативных вариантов является использование **линейного** протокола двухфазной фиксации транзакций, при котором участники могут взаимодействовать друг с другом так, как показано на рис. 20.9, б. При использовании линейной топологии сайты нумеруются в последовательности 1, 2, ..., n, где сайт 1 является координатором, а все остальные сайты — участниками. Протокол двухфазной фиксации транзакций реализуется посредством передачи сообщений по цепочке сайтов в направлении от координатора к участнику n в фазе голосования, а затем в обратном направлении, в фазе принятия решения. В фазе голосования координатор посылает команду PREPARE участнику с номером 2. Этот сайт анализирует ситуацию и передает свое решение сайту с номером 3. Данный сайт комбинирует свое решение с решением сайта 2, после чего направляет полученный результат следующему сайту в цепочке. Когда сообщение достигает последнего участника, он принимает свое решение и формирует глобальное решение, которое посылает в обратном направлении, т.е. предыдущему сайту. Глобальное решение, последовательно пересылаемое по цепочке в обратном направлении, в конце концов достигает координатора. Хотя линейная топология сокращает количество передаваемых сообщений в сравнении с централизованной схемой, последовательная пересылка сообщений исключает возможность распараллеливания вычислений.

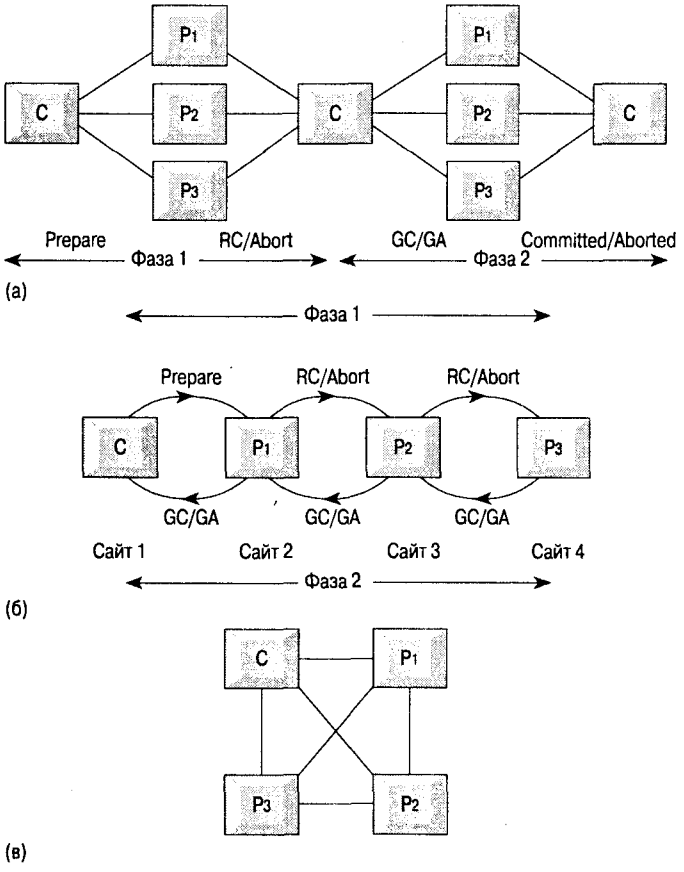


Рис. 20.9. Различные топологии протокола двухфазной фиксации транзакций: а) централизованная; б) линейная; в) распределенная. Здесь: C — координатор; Pi — участник; RC — сообщение Ready_Commit; GC — сообщение Global_Commit; GA — сообщение Global_Abort

Линейная схема двухфазной фиксации транзакций может быть дополнительно улучшена, если процесс голосования выполнять по линейной цепочке в направлении возрастания номеров, а для фазы принятия решения использовать централизованную топологию, что позволит сайту n послать широковещательное сообщение о принятом глобальном решении сразу всем участникам одновременно (Bernstein et al., 1987).

Третий вариант, известный как **распределенный** протокол двухфазной фиксации транзакций, применяет распределенную топологию по схеме, показанной на рис. 20.9, в. Координатор посылает команду PREPARE сразу всем участникам, которые в свою очередь также рассылают сведения о принятых ими решениях на все остальные сайты. Каждый из участников ожидает получения сообщений от всех остальных участников и только после этого принимает решение о фиксации или откате данной транзакции. Такой подход исключает необходимость существования фазы принятия решения, присутствующей в централизованном варианте протокола двухфазной фиксации решений, поскольку каждый из участников может самостоятельно принять решение, согласованное с остальными участниками (Skeen, 1981).

20.4.4. Трехфазная фиксация транзакций (3PL)

Выше уже отмечалось, что двухфазный протокол не является неблокирующим, поскольку при его использовании возможны ситуации, когда некоторый сайт остается в заблокированном состоянии. Например, процесс, зафиксировавший тайм-аут после отправки своего согласия на фиксацию транзакции, но так и не получивший глобального подтверждения от координатора, остается в заблокированном состоянии, если может взаимодействовать только с сайтами, которые также не имеют сведений о принятом глобальном решении. На практике вероятность блокирования процесса достаточно мала, поэтому в большинстве существующих СУРБД используется именно протокол двухфазной фиксации транзакций. Тем не менее был предложен альтернативный неблокирующий протокол, получивший название протокола **трехфазной фиксации транзакций** (Skeen, 1981). Трехфазная фиксация является неблокирующей в отношении отказов сайтов, за исключением случая одновременного отказа всех сайтов. Однако отказы линий связи могут привести к тому, что на различных сайтах будут приняты разные решения, что будет иметь следствием нарушение свойства атомарности глобальной транзакции. Для использования этого протокола необходимо выполнение следующих условий.

- Расчленение сети не должно иметь места.
- По крайней мере один сайт всегда должен быть доступен.
- Самое большее K сайтов сети могут отказать одновременно.

Основная идея протокола трехфазной фиксации состоит в устранении неопределенного периода ожидания, в который попадают участники после подтверждения своего согласия на фиксацию транзакции и до получения от координатора извещения о глобальной фиксации или глобальном откате. В трехфазном протоколе фиксации вводится третья фаза, называемая **предфиксацией**, помещаемая между фазами голосования и принятия глобального решения. После получении результатов голосования от всех участников координатор рассылает глобальное сообщение PRE-COMMIT. Участник, который получил глобальное извещение о предфиксации, знает, что все остальные участники проголосовали за фиксацию результатов транзакции и что со временем сам этот участник определенно выполнит фиксацию транзакции, если не произойдет отказ. Каждый участник подтверждает получение сообщения о предфиксации. После того как координатор получит все эти подтверждения, он рассылает команду глобальной фиксации транзакции. Если некоторый участник потребовал отката транзакции, то обработка этой ситуации выполняется точно так же, как в протоколе двухфазной фиксации.

Модифицированные варианты диаграмм состояния для координатора и участника показаны на рис. 20.10. Как координатор, так и участник по-прежнему попадают в состояние ожидания, однако главная особенность состоит в том, что все *функционирующие* процессы были информированы о глобальном решении зафиксировать транзакцию посредством отправки сообщения PRE-COMMIT еще *до того*, как первый процесс выполнит фиксацию результатов транзакции, что позволяет участникам действовать независимо друг от друга в случае отказа.

20.4.5. Расчленение сетей

При возникновении ситуации расчленения сети поддержание базы данных в согласованном состоянии может оказаться более или менее затруднено, в зависимости от того, имеет место репликация данных или нет. Если репликация данных отсутствует, можно позволить транзакциям продолжать свое выполнение, если они не нуждаются в каких-либо данных, размещенных на сайтах, недоступных для того фрагмента, в котором эти транзакции были инициированы. В противном случае транзакции должны быть переведены в состояние ожидания вплоть до момента, когда связь с требуемыми сайтами будет восстановлена. Если имеет место репликация данных, то процедура существенно усложняется. Ниже мы рассмотрим два примера аномалий, которые могут иметь место при репликации данных в расчлененной сети. Оба примера предусматривают обращение к отношению с обычными банковскими счетами клиентов банка.

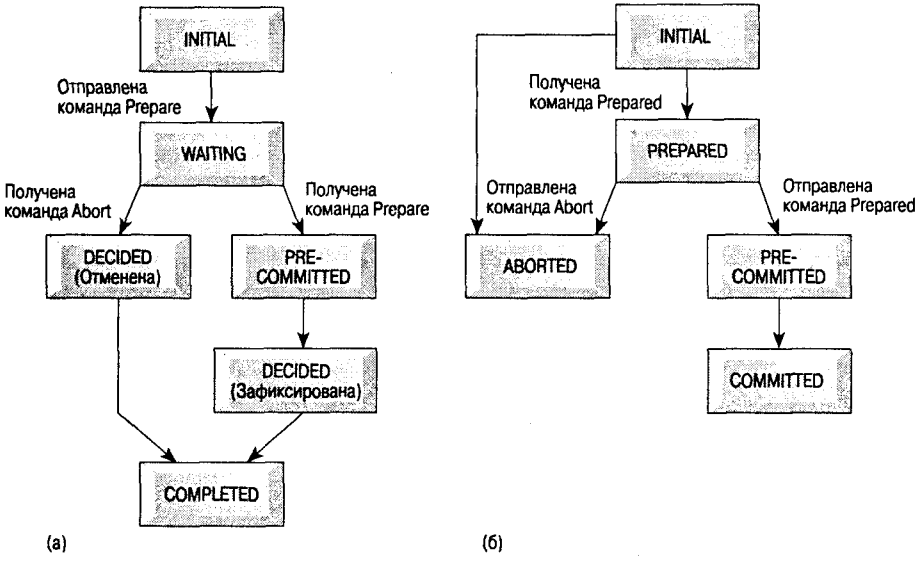


Рис. 20.10. Диаграммы состояния для протокола трехфазной фиксации транзакций: а) координатор; б) участник

Обнаружение обновлений

Успешно завершенные пользователями операции обновления, выполненные в различных фрагментах сети, может быть очень трудно обнаружить, что иллюстрируется примером, приведенным в табл. 20.1. Во фрагменте сети P_1 была выполнена транзакция, в которой со счета bal_x была снята сумма, равная 10 фунтам стерлингов. Во фрагменте P_2 были выполнены две транзакции, в каждой из которых с этого же счета было снято по 5 фунтов стерлингов. Предположим, что исходно в обоих фрагментах сети на счету bal_x находилась сумма 100 фунтов стерлингов. Тогда после выполнения указанных транзакций в каждом из фрагментов сети на счету bal_x останется по 90 фунтов стерлингов. После восстановления целостности сети будет недостаточно проверить текущее значение на счету bal_x, после чего сделать заключение, что согласованность базы данных не нарушена, поскольку в обоих фрагментах сети итоговое значение оказалось одинаковым. На самом деле в этом примере после выполнения всех транзакций остаток на счету bal_x должен равняться 80 фунтам стерлингов.

Таблица 20.1. Пример проблемы обнаружения обновлений

Время	Фрагмент P_1	Фрагмент P_2
t_1	begin_transaction	begin_transaction
t_2	$bal_x = bal_x - 10$	$bal_x = bal_x - 5$
t_3	write(bal_x)	write(bal_x)
t_4	commit	commit
t_5		begin_transaction
t_6		$bal_x = bal_x - 5$
t_7		write(bal_x)
t_8		commit

Поддержание целостности

Успешно завершенные пользователями операции обновления, выполненные в различных фрагментах сети, часто могут вызывать нарушение требований поддержки целостности данных, что иллюстрируется примером, приведенным в табл. 20.2. Предположим, что банк установил ограничение, согласно которому на счету клиентов (атрибут bal_x) не может оставаться отрицательный остаток. Пусть во фрагменте сети P_1 была выполнена транзакция, в которой со счета bal_x была снята сумма 60 фунтов стерлингов, а во фрагменте P_2 была выполнена транзакция, в которой с того же счета bal_x была снята сумма 50 фунтов стерлингов. Предположим, что исходно в обоих фрагментах сети на счету bal_x находилась сумма, равная 100 фунтам стерлингов. Тогда после выполнения указанных транзакций на одном варианте счета bal_x останется 40 фунтов стерлингов, а на другом — 50. Важно отметить, что ни в одном из этих случаев установленное банком ограничение не было нарушено. Однако после восстановления целостности сети сводное значение счета bal_x должно быть принято равным -10 фунтов стерлингов, поскольку каждая из транзакций была зафиксирована в своем фрагменте сети и уже не может быть отменена. В результате установленные требования поддержки целостности данных будут нарушены.

Таблица 20.2. Пример проблемы поддержания целостности данных

Время	P_1	P_2
t_1	begin_transaction	begin_transaction
t_2	$bal_x = bal_x - 60$	$bal_x = bal_x - 50$
t_3	write(bal_x)	write(bal_x)
t_4	commit	commit

Работа в расчлененной сети требует принятия определенного компромисса между доступностью данных и корректностью их использования (Davidson, 1984; Davidson et al., 1985). Абсолютную корректность проще всего достичь, если в случае расчленения сети полностью запретить обработку каких-либо реплицируемых данных. С другой стороны, доступность данных в той же ситуации будет максимальной, если на обработку реплицируемых данных не накладывать никаких ограничений.

В общем случае невозможно создать неблокирующий, соблюдающий свойство атомарности, протокол фиксации транзакций для случая произвольного расчленения сети (Skeen, 1981). Поскольку протоколы восстановления и управления параллельностью тесно связаны между собой, выбор метода восстановления, используемого в случае расчленения сети, будет непосредственно зависеть от применяемой в системе стратегии управления параллельностью. Соответственно методы восстановления могут классифицироваться как пессимистические и оптимистические.

Пессимистические протоколы

В пессимистических протоколах предпочтение отдается сохранению целостности базы данных, а не обеспечению доступности этих данных пользователям, поэтому в отдельных фрагментах сети не допускается выполнение любых транзакций, для которых не существует гарантий, что в результате их выполнения целостность базы данных не будет нарушена. Подобные протоколы используют пессимистические алгоритмы управления параллельностью, например метод двухфазной блокировки с первичными копиями или метод блокирования большинства, обсуждавшиеся в разделе 20.2. Процедура восстановления в этом случае совсем проста, поскольку любые выполненные обновления касались информации, доступной только в отдельном фрагменте. После воссоединения сети в единое целое для восстановления целостности базы данных достаточно просто распространить сведения обо всех выполненных в различных фрагментах сети изменениях на все прочие сайты.

Оптимистические протоколы

В оптимистических протоколах, наоборот, предпочтение отдается доступности данных, даже за счет возможной потери целостности базы данных. Кроме того, в этом случае используются оптимистические протоколы управления параллельностью, позволяющие независимо выполнять любые обновления в каждом из фрагментов сети. В результате после воссоединения сети в единое целое весьма вероятен переход базы данных в несогласованное состояние.

Для выявления нарушений целостности базы данных может использоваться **граф предшествования**, содержащий сведения о взаимосвязях элементов данных. Граф предшествования подобен обсуждавшемуся выше графу ожидания. Он содержит сведения о том, какие элементы данных считывала и записывала каждая из выполнявшихся транзакций. Пока сеть находится в расчлененном состоянии, обновления выполняются без каких-либо ограничений, но для каждого из фрагментов сети строится собственный граф предшествования. После воссоединения сети графы предшествования всех фрагментов сливаются. Нарушение целостности базы данных имеет место в том случае, если сводный граф предшествования содержит петли. Способ устранения нарушения целостности зависит от семантики выполнявшихся транзакций, поэтому в общем случае менеджер восстановления не сможет восстановить целостность базы данных без вмешательства пользователей.

20.5. Модель распределенной обработки транзакций X/Open

Open Group представляет собой независимый международный консорциум пользователей, разработчиков программ и производителей оборудования, задача которого состоит в содействии созданию жизнеспособной глобальной информационной инфраструктуры. Он был создан в феврале 1996 года в результате слияния компании X/Open Company Ltd (основанной в 1984 году) и фонда Open Software Foundation (основанного в 1988 году). Компания X/Open создала рабочую группу Distributed Transaction Processing (DTP) Working Group, целью которой является разработка и модернизация программных интерфейсов, необходимых для обработки распределенных транзакций. Однако в то время системы обработки транзакций понимались как законченные функциональные среды, включавшие все, начиная от определений экранов пользователей и вплоть до реализации баз данных. Вместо того чтобы попытаться создать пакет стандартов для каждой из отдельных областей, группа сосредоточилась на тех элементах системы обработки транзакций, которые призваны обеспечивать основные свойства транзакций (ACID-свойства), обсуждавшиеся в разделе 17.1.1. В выпущенном группой X/Open DTP стандарте определяются три взаимодействующих между собой компонента: приложение, менеджер транзакций (TM) и менеджер ресурсов (RM).

Менеджером ресурсов может быть любая подсистема, управляющая используемыми в транзакциях данными, например система управления базой данных или файловая система в совокупности с менеджером сеансов. Менеджер транзакций отвечает за определение границ транзакции, т.е. за определение, какие операции являются элементами данной транзакции. Кроме того, он отвечает за присвоение каждой из транзакций некоторого уникального идентификатора, используемого всеми компонентами, а также координирует работу остальных компонентов с целью определения результатов выполнения транзакции. Для организации завершения работы распределенной транзакции менеджер транзакций может координировать свою работу с другими менеджерами транзакций. Начиная выполнение некоторой транзакции, приложение прежде всего обращается к менеджеру транзакции, а затем к менеджерам ресурсов с целью выполнения манипуляций данными, необходимых для реализации логики их обработки. По окончании обработки данных приложение вновь обращается к менеджеру транзакций с требованием завершения данной транзакции. Для координации выполнения транзакции менеджер транзакции взаимодействует с менеджерами ресурсов.

В модели X/Open дополнительно определено несколько интерфейсов, показанных на рис. 20.11. Приложение может использовать интерфейс TX для взаимодействия с менеджером транзакций. Интерфейс TX включает вызовы функций, предназначенных для определения границ транзакции (иногда их называют *ограничителями транзакции*), а также для фиксации или отмены транзакции. Сведения об обрабатываемой в транзакции информации менеджер транзакций получает от менеджера ресурсов через интерфейс XA. Наконец, приложение может взаимодействовать непосредственно с менеджером ресурсов через обычный программный интерфейс, например интерфейс языка SQL или метод доступа ISAM.

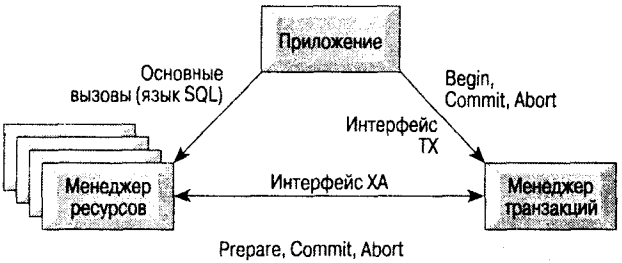


Рис. 20.11. Интерфейсы стандарта X/Open

Например, рассмотрим приведенный ниже фрагмент некоторого приложения:

```
tx_begin();
    EXEC SQL UPDATE Staff SET salary = salary*1.05 WHERE Position = 'Manager';
    EXEC SQL UPDATE Staff SET salary = salary*1.04 WHERE Position <> 'Manager';
tx_commit();
```

При обращении приложения к функции tx_begin() интерфейса Call Level Interface (CLI) менеджер транзакций организует начало новой транзакции и присваивает ей уникальный идентификатор. При этом менеджер транзакций с помощью функций интерфейса XA информирует о начале новой транзакции менеджер ресурсов, которым в данном случае является сервер базы данных SQL. Получив извещение от менеджера транзакций, менеджер ресурсов предполагает, что любые поступающие от приложения вызовы будут являться частью данной транзакции, — в данном случае это два SQL-оператора обновления данных. Наконец, приложение заканчивает выполнение транзакции и вызывает функцию tx_commit(), после чего менеджер транзакций посылает менеджеру ресурсов указание зафиксировать результаты выполнения данной транзакции. Если приложение одновременно работает более чем с одним менеджером ресурсов, то в этом случае для синхронизации операций фиксирования результатов транзакции каждым менеджером ресурсов менеджер транзакции воспользуется двухфазным протоколом фиксации.

В распределенной среде описанная выше модель взаимодействия должна быть модифицирована так, чтобы позволить транзакции включать несколько отдельных субтранзакций, каждая из которых будет выполняться на удаленном сайте в удаленной базе данных. Схема модели X/Open DTP для распределенной среды представлена на рис. 20.12. В этом варианте модели приложение взаимодействует с особым типом менеджера ресурсов, называемым менеджером передачи данных (Communication manager — CM). Как и все остальные типы менеджеров ресурсов, менеджер передачи данных получает сведения о выполняемых транзакциях от менеджера транзакций. Приложение взаимодействует с менеджером передачи данных через один из типовых интерфейсов. В этом случае необходимо наличие двух механизмов — удаленного запуска и поддержки распределенных транзакций. Удаленный запуск может быть организован с помощью стандартизованных ISO механизмов ROSE (Remote Operations Server) или RPC (Remote Procedure Call). Для координации выполнения распределенной транзакции стандарт X/Open определяет коммуникационный протокол OSI-TP (Open Systems Interconnection Transaction Processing).

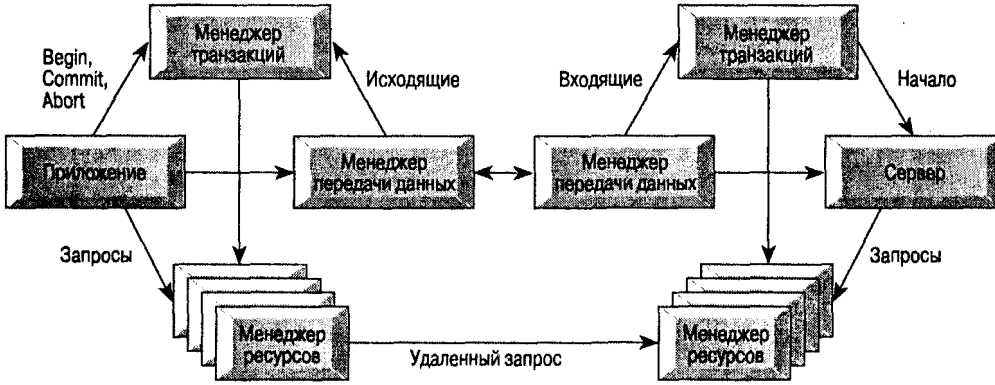


Рис. 20.12. Интерфейсы модели X/Open в распределенной среде

Модель X/Open DTP поддерживает выполнение не только плоских транзакций, но также цепочечных и вложенных транзакций (см. раздел 17.4. В случае вложенных транзакций при отмене любой из субтранзакций будет отменена и вся глобальная транзакция.

Модель X/Open нашла широкую поддержку у производителей программного обеспечения. Несколько независимых компаний выпустили на рынок пакеты менеджеров транзакций, поддерживающих интерфейс TX, а многие разработчики коммерческих СУБД реализовали в своих продуктах поддержку протокола XA. Примерами подобных систем являются пакет Encina компании Transarc, СУБД Tuxedo, Oracle, Informix и SQL Server.

20.6. Серверы репликации

Как уже отмечалось в разделе 19.1.2, в настоящее время в эксплуатации находится несколько функционирующих прототипов распределенных СУБД, а также несколько систем специального назначения, что позволило уточнить требования к используемым протоколам и выявить другие важные проблемы. Однако к моменту написания этой книги распределенные СУБД общего назначения еще не получили широкого распространения. Более популярным решением в настоящее время является репликация данных — копирование и сопровождение данных на нескольких серверах. Каждый крупный поставщик СУБД реализует в своих продуктах тот или иной вариант механизма репликации, кроме того, существует много сторонних разработчиков, предлагающих собственные альтернативные решения для организации репликации данных. Использование серверов репликации является альтернативным и потенциально более простым подходом к созданию распределенных систем.

20.6.1. Основные концепции репликации данных

Проще всего репликацию можно определить как процесс генерации и воспроизведения нескольких копий данных, размещаемых на одном или нескольких сайтах. 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. В этом разделе мы рассмотрим несколько основных концепций, имеющих отношение к механизмам репликации, включая ожидаемый набор функциональных возможностей и особенности определения владельца данных. Однако прежде всего следует обсудить вопрос, где должны обновляться реплицируемые данные.

Синхронная и асинхронная репликация

Протоколы обновления реплицируемых данных, с которыми мы познакомились в предыдущих разделах этой главы, 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 (см. раздел 20.4.3). 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), выше было показано, что ему свойственны определенные недостатки. В частности, транзакция не сможет быть завершена, если один из сайтов 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 *асинхронного*. 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 организаций, чья деятельность допускает работу с копией данных, необязательно точно синхронизованной на текущий момент.

Функциональность

Мы ожидаем, что в качестве базового уровня служба репликации распределенных данных должна быть способна копировать данные из одной базы данных в другую синхронно или асинхронно. Кроме того, существует множество других функций, которые должны поддерживаться, включая следующие.

- *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. Как уже отмечалось в разделе 19.1.3, 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.

Владение данными

Владение данными определяет, какому из сайтов будет предоставлена привилегия обновления данных. Основными типами схем владения являются варианты **"ведущий/ведомый"**, **"рабочий поток"** и **"повсеместное обновление"**. Последний вариант иногда называют *одноранговой*, или *симметричной репликацией*.

Схема владения „ведущий/ведомый"

При организации владения данными по схеме "ведущий/ведомый" асинхронно реплицируемые данные принадлежат одному из сайтов, называемому *ведущим*, или *первичным*, и могут обновляться только на нем. Здесь можно провести аналогию между издателем и подписчиками. Издатель (ведущий сайт) публикует свои данные. Все остальные сайты 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 сайт, 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. В компании *DreamHome* может собираться информация обо всех сдаваемых в аренду и продаваемых объектах недвижимости со сведениями об их арендаторах и покупателях. Затем эти данные могут анализироваться различными способами с целью выявления существующих тенденций спроса и определения категорий клиентов, приобретающих те или иные виды собственности в различных регионах страны.
- *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. Например, описания продукции и прайс-лист компании могут готовиться на центральном сайте компании, после чего доступные только для чтения копии этих документов реплицируются на удаленные сайты всех ее отделений. Этот вариант репликации данных показан на рис. 20.13, 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. Например, это могут быть сведения об объектах недвижимости, которыми занимаются в каждом из отделений компании *DreamHome*. Доступные только для чтения копии этих данных реплицируются на центральный сайт, где помещаются (консолидируются) в единый сводный набор данных. Этот вариант репликации данных показан на рис. 20.13, б.
- *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 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 (например, сведения о новом заказчике или о новом заказе), обрабатываются сходным образом.

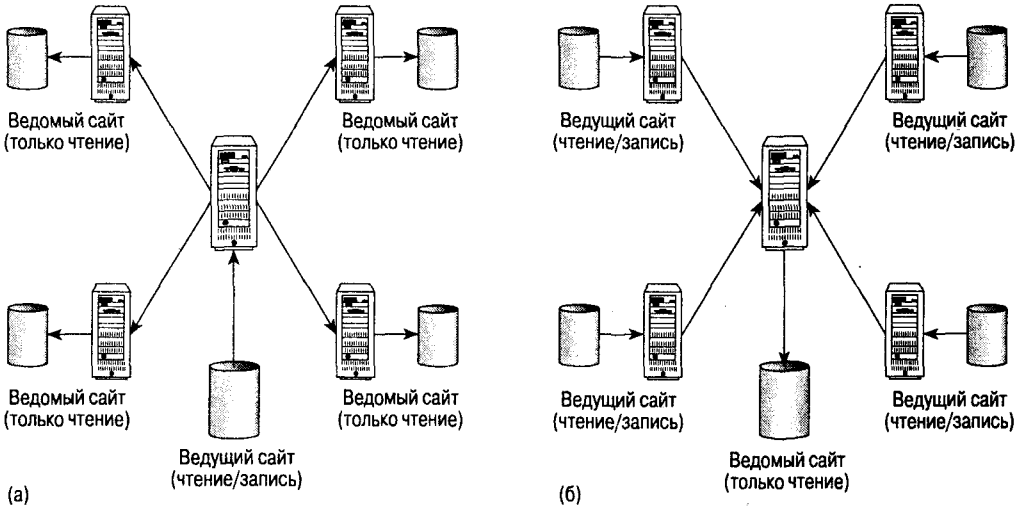


Рис. 20.13. Владение данными по схеме "ведущий/ведомый": а) распределение данных; б) консолидация данных

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**.

В распределенной СУБД компании *DreamHome* каждому отделению может быть позволено владеть своими собственными горизонтальными фрагментами таблиц Property_for_Rent, Renter и Lease_Agreement. Сайт центрального правления компании может подписаться на данные, принадлежащие каждому из отделений компании, после чего консолидировать поступившие копии в доступные только для чтения сводные таблицы с информацией об объектах недвижимости, клиентах и заключенных соглашениях по всей компании в целом.

Схема владения „рабочий поток"

Как и в случае схемы "ведущий/ведомый", в этой модели 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. 20.14.

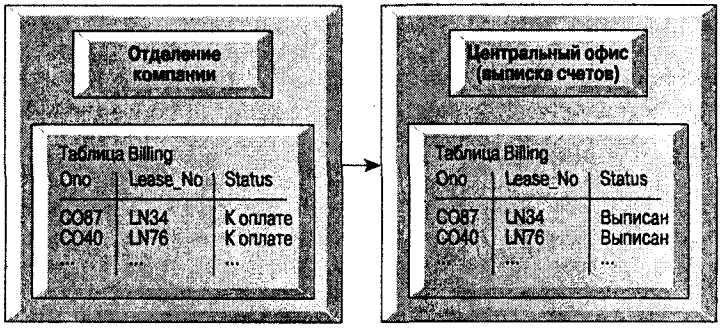


Рис. 20.14. Схема владения "рабочий поток"

Схема владения „повсеместное обновление" (симметричная репликация)

У двух предыдущих моделей есть одно общее свойство: в любой заданный момент времени только один сайт имеет право обновлять данные. Всем остальным сайтам доступ к репликам данным будет разрешен только для чтения. В некоторых случаях это ограничение оказывается слишком жестким. Схема владения с повсеместным обновлением создает равноправную среду, в которой множество сайтов имеют одинаковые права на обновление реплицируемых данных. В результате локальные сайты получают возможность работать автономно, даже в тех случаях, когда другие сайты недоступны. Например, в компании *DreamHome* может быть принято решение организовать "горячую линию", с помощью которой потенциальные клиенты, желающие приобрести или взять в аренду объекты недвижимости, смогут позвонить по бесплатной междугородной телефонной линии и описать интересующий их тип объектов, договориться о просмотре объектов или вообще выполнить любые действия, которые были бы возможны в случае посещения клиентом отделения компании. Подобная телефонная служба была организована в каждом отделении компании. Все звонки автоматически переправляются в отделение компании, ближайшее к месту нахождения клиента. Например, если клиент интересуется объектами недвижимости в Лондоне, но звонит из Глазго, то его вызов направляется в отделение компании, расположенное в Глазго. Система телекоммуникации поддерживает баланс нагрузки, поэтому если телефонная линия в Глазго будет занята, то вызов клиента переправят в Эдинбург. Каждый центр обработки звонков клиентов должен иметь доступ к данным, принадлежащим всем остальным отделениям компании, и в случае необходимости иметь право обновлять записи, реплицированные с других сайтов, как показано на рис. 20.15.

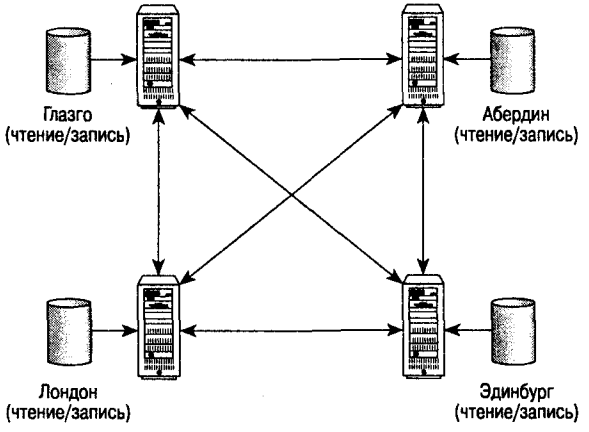


Рис. 20.15. Схема вычислительной среды с симметричной репликацией

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. Подробнее об этом речь пойдет чуть позже.

Сохранение целостности транзакций

Первые попытки реализации механизма репликации по самой своей сути не предусматривали сохранения целостности транзакций. Данные копировались без сохранения свойства атомарности транзакций, что потенциально могло привести к утрате целостности распределенных данных. Этот подход проиллюстрирован на рис. 20.16, а. 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.

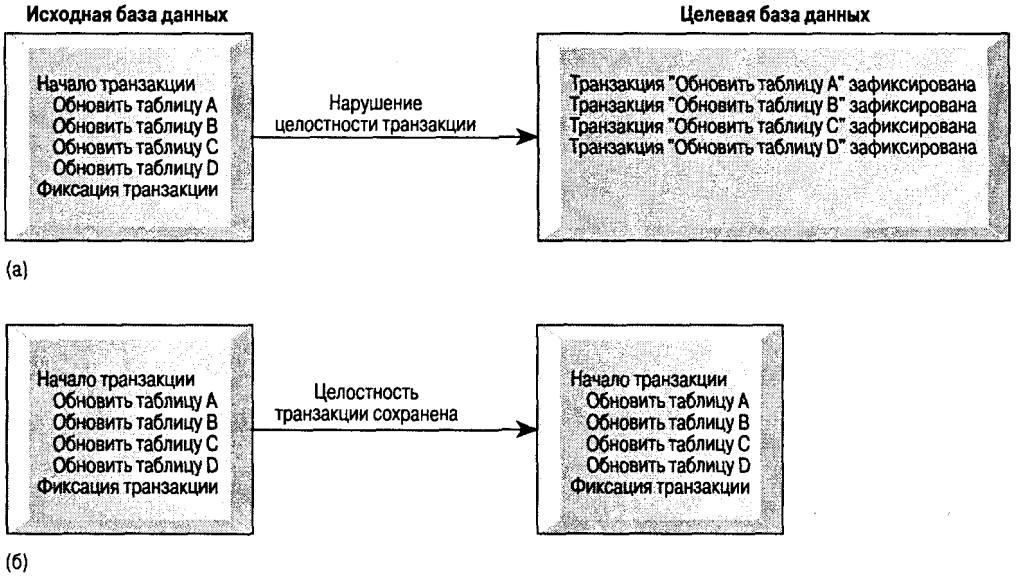


Рис. 20.16. Репликация транзакций: а) без соблюдения свойства атомарности; б) с соблюдением атомарности транзакций

В противоположность этому, на рис. 20.16, б показан пример репликации с сохранением структуры транзакции в исходной базе данных.

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, например ежедневно в 23.00. В СУБД Oracle 8 описать моментальный снимок со сведениями о персонале отделения компании с номером 'B5' можно с помощью следующих операторов:

```
CREATE SNAPSHOT local_staff
REFRESH FAST
START WITH sysdate NEXT sysdate + 7
AS SELECT * FROM Staff@Staff_Master_Site WHERE bno = 'B5';
```

В этом примере предложение SELECT определяет те строки ведущей таблицы (расположенной на сайте Staff_Master_Site), которые должны подвергаться репликации. Предложение REFRESH определяет конкретный механизм СУБД Oracle, который должен применяться при обновлении описываемого моментального снимка. Возможны три различных варианта: FAST, COMPLETE и FORCE. Если параметр REFRESH имеет значение FAST, в моментальный снимок из ведущей таблицы отсылаются только те строки, которые были изменены. Если значение параметра равно COMPLETE, вся информация в моментальном снимке формируется заново. Значение параметра FORCE позволяет СУБД Oracle самостоятельно выбрать оптимальный вариант пересылки данных (FAST или COMPLETE). Предложение START WITH указывает, что создаваемый моментальный снимок должен обновляться раз в семь дней, начиная с текущего. Если одновременно должны быть обновлены два или больше моментальных снимков (например, из соображений сохранения целостности между двумя таблицами), СУБД Oracle позволяет определить группу обновляемых моментальных снимков.

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 источником 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. В частности, в СУБД Oracle для поддержания копии таблицы Staff на некотором удаленном сайте (адресуемом с помощью связи с именем Staff_Duplicate_Link) можно использовать следующую процедуру:

```
CREATE TRIGGER staff_after_ins_row
BEFORE INSERT ON Staff
FOR EACH ROW
BEGIN
    INSERT INTO Staff_Duplicate@Staff_Duplicate_Link
    VALUES (:new.Sno, :new:FName, :new:LName, :new.Address, :new:Tel_No,
            :new.Position, :new:Sex, :new.DOB, :new:Salary, :new.NIN, :new:Bno);
END;
```

Этот триггер будет вызываться при вставке в таблицу Staff каждой новой записи.

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 сайтов 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.

- *Самая ранняя или самая поздняя временная отметка.* Изменяются соответственно данные с самой ранней или самой поздней временной отметкой.
- *Приоритеты сайтов.* Применяется обновление, поступившее с сайта 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, например `salary = salary + x`.
- *Минимальное или максимальное значение.* Применяются обновления, соответствующие столбцу с минимальным или максимальным значением.
- *По решению пользователя.* АБД 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.

20.7. Оптимизация распределенных запросов

В главе 18, "Обработка запросов" мы обсудили вопросы обработки и оптимизации запросов в централизованных реляционных СУБД. Рассматривались два метода оптимизации запросов:

- использование **эвристических правил** для переупорядочивания операций запроса;
- сравнение различных стратегий на основе их относительных оценок и выбор стратегии с минимальным использованием системных ресурсов.

Поскольку доступ к данным на диске осуществляется во много раз медленнее, чем доступ к данным в оперативной памяти, оценка стоимости различных стратегий выполнения запроса в централизованных СУБД устанавливалась нами по количеству требуемых дисковых операций. Именно этот единственный показатель учитывался в главе 18 при определении оценки стоимости. Однако в распределенной среде при сравнении различных стратегий во внимание следует принимать и скорость передачи данных по имеющимся сетевым соединениям. Как уже упоминалось выше, в разделе 19.5.3, некоторые соединения в глобальных вычислительных сетях (ГВС) могут иметь скорость передачи данных, равную лишь нескольким килобайтам в секунду. Если для конкретной сети известно, что она имеет топологию ГВС, то можно игнорировать все другие оценки стоимости, за исключением стоимости передачи данных по глобальной сети. Локальные вычислительные сети (ЛВС) обычно имеют существенно более высокую скорость передачи данных, чем ГВС, однако она все же ниже скорости доступа к данным на жестком диске. В обоих случаях по-прежнему может применяться наше общее правило оптимизации запросов, требующее минимизировать размер всех выполняемых операций реляционной алгебры и выполнять любые унарные операции до выполнения бинарных.

20.7.1. Преобразование распределенных запросов

В главе 18 мы сначала представляли оптимизируемые запросы в виде дерева реляционной алгебры, после чего это дерево с помощью правил преобразования приводилось в эквивалентную форму, имеющую более высокие показатели скорости обработки. При работе с распределенными запросами на начальной стадии применяется тот же самый подход. Однако при применении эвристических стратегий оптимизации, обсуждавшихся в разделе 18.3.2, к глобальным запросам следует принять во внимание наличие распределенности данных в сети. С этой целью мы заменим расположенные в листьях дерева глобальные отношения их **алгоритмами реконструкции**. Последние представляют собой операции реляционной алгебры, необходимые для реконструкции глобальных отношений из фрагментов, на которые эти отношения разбиты. При горизонтальной фрагментации алгоритм реконструкции будет состоять из операций объединения. В случае вертикальной фрагментации этот алгоритм будет включать операции соединения. Дерево реляционной алгебры запроса, дополненное требуемыми алгоритмами реконструкции, в некоторых случаях называют *общим деревом реляционной алгебры*. Следовательно, для построения упрощенного и оптимизированного запроса мы будем использовать *методы упрощения* общих деревьев реляционной алгебры. Конкретный тип применяемого метода упрощения будет зависеть от особенностей фрагментации данных в системе. Ниже мы рассмотрим методы упрощения для следующих типов фрагментации:

- простая горизонтальная фрагментация;
- вертикальная фрагментация;
- производная фрагментация.

Методы упрощения для простой горизонтальной фрагментации

Для простой горизонтальной фрагментации мы рассмотрим две возможные методики: сокращение операций выборки и упрощение операций соединения. В первом варианте утверждается, что если предикат операции выборки противоречит определению фрагмента, то результирующая таблица окажется пустой, поэтому данную операцию выборки можно исключить. Во втором случае предлагается сначала применить правило преобразования, по которому операция соединения может быть заменена операцией объединения результатов частичных соединений:

$$(R1 \cup R2) \bowtie R3 = (R1 \bowtie R3) \cup (R2 \bowtie R3)$$

Затем анализируется, не является ли каждая частичная операция соединения бесполезной и нельзя ли ее безболезненно исключить. Соединение будет бесполезно в том случае, если предикаты используемых фрагментов не перекрываются. Это правило преобразования очень важно в распределенной СУБД, поскольку позволяет реализовать соединение двух отношений как объединение частичных соединений, каждое из которых может выполняться параллельно с другими. Проиллюстрируем применение двух приведенных выше правил на примере.

Пример 20.2. Упрощение в случае простой горизонтальной фрагментации

Составьте список квартир, сданных в аренду, с указанием сведений о соответствующем отделении компании.

Этот запрос может быть представлен следующим SQL-оператором:

```
SELECT *
FROM branch b, property_for_rent p
WHERE b.bno = p.bno AND p.type = 'Flat';
```

Теперь предположим, что отношения Property_for_Rent и Branch разбиты на горизонтальные фрагменты по следующему правилу:

P_1: $\sigma_{bno='B3' \wedge type='House'}(Property\_for\_Rent)$ B_1: $\sigma_{bno='B3'}(Branch)$

P_2: $\sigma_{bno='B3' \wedge type='Flat'}(Property\_for\_Rent)$ B_2: $\sigma_{bno!='B3'}(Branch)$

P_3: $\sigma_{bno!='B3'}(Property\_for\_Rent)$

Общее дерево реляционной алгебры для этого запроса показано на рис. 20.17, а. Если упростить операции выборки и объединения, то получим дерево реляционной алгебры, показанное на рис. 20.17, б. Это дерево получено после выяснения того факта, что следующие ветви исходного дерева являются избыточными (т.е. не вносят в результат ни одного кортежа):

$$\sigma_{type='Flat'}(P_1) = \sigma_{type='Flat'}(\sigma_{bno='B3' \wedge type='House'}(Property\_for\_Rent)) = \varnothing$$

Более того, поскольку предикат операции выборки является подмножеством определения фрагмента P_2, эта операция выборки не нужна. Если теперь упростить операции соединения и объединения, получим дерево реляционной алгебры, показанное на рис. 20.17, в. Поскольку второе и третье соединения не вносят в результат ни одного кортежа, их можно исключить. В результате мы получим окончательный вариант сокращенного дерева запроса, показанный на рис. 20.17, г.

Методы упрощения для вертикальной фрагментации

Упрощение в случае вертикальной фрагментации предусматривает удаление тех вертикальных фрагментов, которые не имеют общих атрибутов с условием проекции, за исключением ключа отношения.

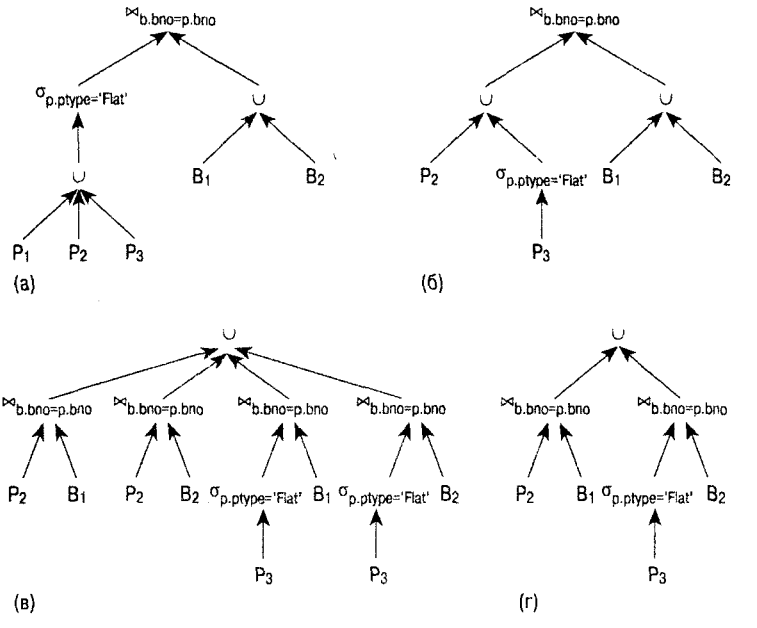


Рис. 20.17. Дерево реляционной алгебры для запроса из примера 20.2: а) общее дерево запроса; б) дерево, полученное после удаления операции выборки; в) дерево, полученное поле замены операции соединения операцией объединения; г) окончательный вариант упрощенного дерева запроса

Пример 20.3. Упрощение в случае вертикальной фрагментации

Составьте список имен всех сотрудников компании.

Этот запрос может быть представлен следующим SQL-оператором:

```
SELECT fname, lname
FROM staff;
```

В этом примере мы воспользуемся той же схемой фрагментации отношения Staff, которую использовали в примере 19.3:

S_1: $\Pi_{sno,\ position,\ sex,\ dob,\ salary,\ nin}(Staff)$

S_2: $\Pi_{sno,\ fname,\ lname,\ address,\ tel\ no,\ bno}(Staff)$

Общее дерево реляционной алгебры для этого запроса показано на рис. 20.18, а. После применения правила подстановки операций проекции и соединения операция проекции для фрагмента S_1 оказывается излишней, поскольку требуемые атрибуты FName и LNane в этом фрагменте отсутствуют. Упрощенное дерево запроса показано на рис. 20.18, б.

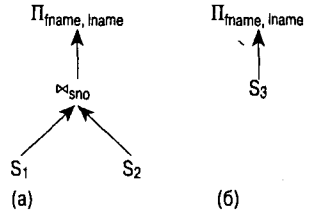


Рис. 20.18. Дерево реляционной алгебры для запроса из примера 20.3: а) общее дерево запроса; б) упрощенное дерево запроса

Методы упрощения для производной фрагментации

Упрощение в случае производной фрагментации также предусматривает применение правила преобразования, по которому операция соединения может быть заменена операцией объединения результатов частичных соединений. В этом случае используется знание того, что фрагментация одного отношения выполнена на основе фрагментации другого, поэтому после замены операций некоторые из частичных соединений могут оказаться избыточными.

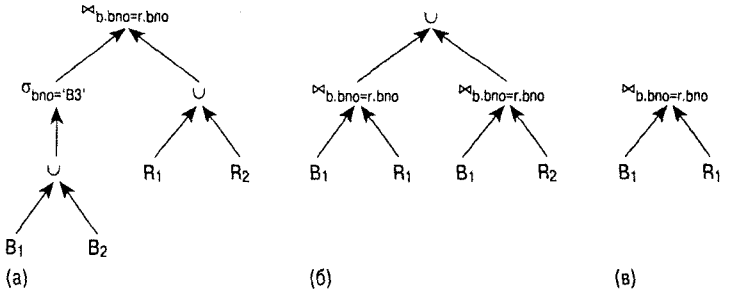


Рис. 20.19. Дерево реляционной алгебры для запроса из примера 20.4: а) общее дерево запроса; б) упрощенное дерево запроса после подстановки операций соединения и объединения; в) окончательный вариант упрощенного дерева запроса

Пример 20.4. Упрощение в случае производной фрагментации

Составьте список арендаторов, зарегистрированных в отделении компании с номером 'B3', с указанием сведений об этом отделении.

Этот запрос может быть представлен следующим SQL-оператором:

```
SELECT *
FROM branch b, renter r
WHERE b.bno = r.bno AND b.bno = 'B3';
```

Предположим, что фрагментация отношения Branch выполнена так, как показано в примере 20.2, а фрагментация отношения Renter является производной от фрагментации отношения Branch:

$B_1 = \sigma_{bno='B3'}$ (Branch) $\quad B_2 = \sigma_{bno!='B3'}$ (Branch)

R_i = Renter $\triangleright_{bno} B_i \quad i = 1, 2$

Общее дерево реляционной алгебры для этого запроса показано на рис. 20.19, а. После применения правила подстановки операций выборки и объединения операция выборки для фрагмента B_2 оказывается излишней и может быть исключена.

Сама операция выборки также может быть исключена, поскольку фрагмент B_1 определен для отделения компании с номером 'B3'. Если теперь выполнить подстановку операций соединения и объединения, будет получено дерево, показанное на рис. 20.19, б. Вторая операция соединения между фрагментами B_1 и R_2 даст пустой результат и может быть безболезненно удалена. Окончательный вид упрощенного дерева показан на рис. 20.19, в.

20.7.2. Операции соединения в распределенной среде

Операция соединения является одной из наиболее дорогостоящих операций реляционной алгебры. Один из подходов, который может использоваться при оптимизации распределенных запросов, состоит в замене соединений комбинацией операций полусоединения (см. раздел 3.4.1). Операция полусоединения имеет важное свойство: она позволяет сократить размер обрабатываемого отношения. Когда наиболее дорогостоящим компонентом является время передачи данных, операция полусоединения может оказаться весьма полезной для ускорения обработки распределенных операций соединения, поскольку позволит сократить объем данных, подлежащих пересылке между сайтами.

Например, предположим, что на сайте S_2 требуется получить результат операции соединения $R_1 \rhd\lhd_x R_2$, где R_1 и R_2 являются фрагментами, сохраняемыми на сайтах S_1 и S_2 соответственно. Фрагменты R_1 и R_2 определены на атрибутах $A=(a_1, a_2, \ldots, a_n)$ и $B=(b_1, b_2, \ldots, b_m)$ соответственно. Данное выражение можно переписать с использованием операции полусоединения:

$$R_1 \rhd\lhd_X R_2 = (R_1 \rhd_X R_2) \rhd\lhd R_2$$

Таким образом, требуемый результат можно получить с помощью следующих манипуляций.

1. Вычислить $R' = \Pi_x(R_2)$ на сайте S_2 (атрибуты, необходимые для соединения на сайте S_1).
2. Передать R' на сайт S_1.
3. Вычислить $R'' = R_1 \rhd x\ R'$ на сайте S_1.
4. Передать R'' на сайт S_2.
5. Вычислить $R'' \rhd\lhd_x R_2$ на сайте S_2.

Использование полусоединения имеет смысл только в том случае, если лишь незначительная часть кортежей отношения R_1 будет участвовать в операции соединения отношений R_1 и R_2. Если в соединении будет участвовать большинство кортежей отношения R_1, то лучше сохранить операцию соединения, поскольку при использовании полусоединения дополнительно потребуется передача проекции соединяемых атрибутов. Подробную информацию о методах использования операции полусоединения заинтересованный читатель найдет в работе Бернштейна и Чуи (Bernstein and Chiu, 1981). Следует отметить, что операции полусоединения не используются в какой-либо из ведущих коммерческих СУРБД.

Резюме

- Цели, стоящие при обработке распределенных транзакций, не отличаются от преследуемых при обработке транзакций в централизованных системах. Однако достичь этих целей в случае СУРБД сложнее, поскольку требуется обеспечить неделимость не только глобальной транзакции в целом, но и всех ее субтранзакций.
- Если графики выполнения транзакций на каждом из сайтов упорядочены, то **глобальный график** (объединение всех локальных графиков) также будет упорядочен с тем же типом, что и локальные графики. Это означает, что все суб-

транзакции будут появляться в одном и том же порядке в эквивалентных последовательных графиках на всех сайтах.

- Для обеспечения распределенной упорядоченности могут использоваться два метода: с использованием **блокирования** или **временных отметок. Двухфазный протокол блокирования** требует, чтобы в транзакции все требуемые блокировки были установлены до отмены хотя бы одной из них. Протоколы двухфазной блокировки могут использовать централизованную схему, схему с первичными копиями и распределенную схему. Кроме того, может использоваться метод блокирования большинства. При использовании **временных отметок** транзакции упорядочиваются таким образом, что наиболее старые транзакции в случае конфликта получают преимущество.
- Выявление ситуаций **распределенной взаимной блокировки** требует слияния локальных графов ожидания с последующим анализом глобального графа на наличие петель. При обнаружении петли для ее устранения одна или более транзакций должны быть отменены и перезапущены. В распределенных СУБД существуют три общие схемы выявления распределенной взаимной блокировки: **централизованная, иерархическая и распределенная.**
- Специфическими причинами отказов в распределенной среде являются потеря сообщений, отказ линий связи и сетевых соединений, отказ отдельных сайтов и расчленение сети. Для организации восстановления на каждом из сайтов создается локальный журнал регистрации событий, который используется для отката и повторного выполнения транзакций, необходимого при восстановлении после отказа.
- **Двухфазный протокол фиксации транзакций** состоит из этапов голосования и принятия глобального решения. На первом этапе координатор опрашивает участников об их готовности к фиксации транзакции. Если хотя бы один из участников проголосует за отмену транзакции, глобальная транзакция и все ее локальные субтранзакции будут отменены. Глобальная транзакция может быть зафиксирована только в том случае, если *все* участники проголосовали за фиксацию результатов. При использовании двухфазного протокола фиксации транзакций в среде, подверженной отказам, некоторые сайты могут остаться заблокированными.
- **Трехфазный протокол фиксации транзакций** является неблокирующим. Он предполагает рассылку координатором транзакции между этапами голосования и принятия решения дополнительных сообщений в адрес всех участников транзакции. Эти сообщения указывают участникам на необходимость перейти в состояние предфиксации транзакции.
- Модель **X/Open DTP** представляет собой один из вариантов архитектуры обработки распределенных транзакций, построенной на применении протокола OSI-TP и двухфазного протокола фиксации транзакций. В этой модели определяются прикладные программные интерфейсы и методы взаимодействия между приложениями, запускающими транзакции, менеджерами транзакций, менеджерами ресурсов и менеджерами передачи данных.
- **Репликацией** называется процесс генерации и воспроизведения нескольких копий данных на одном или более сайтов. Это очень важный механизм, поскольку с его помощью организации могут предоставлять своим пользователям доступ к актуальной информации там и тогда, когда это им требуется. Использование репликации позволяет достичь многих преимуществ, включая повышение производительности (в тех случаях, когда централизованные ресурсы оказываются перегруженными), повышение надежности хранения и доступности данных, а также обеспечение поддержки мобильных пользователей и хранилищ данных, предназначенных для систем поддержки принятия решений.

- Хотя асинхронное распространение изменений нарушает принцип независимости распределенных данных, на практике этот метод оказывается удобным компромиссом между требованиями поддержки целостности и доступности данных. Он может оказаться более удобным для организаций, которые допускают работу с копиями данных, необязательно синхронизованными на текущий момент.
- **Моделями владения** для реплицируемых данных могут быть модели "ведущий/ведомый", "рабочий поток" и "повсеместное обновление". В первых двух моделях все копии данных доступны только для чтения. В последнем варианте модели обновления могут независимо вноситься в любую из существующих копий, поэтому для сохранения целостности данных в системе необходимо использование механизма выявления и разрешения возможных конфликтов обновления данных.
- Типичными механизмами организации репликации являются использование моментальных снимков таблиц и триггеров базы данных. Распространение сведений об обновлениях данных может проводиться с соблюдением и без соблюдения целостности транзакций.
- Когда основным компонентом, определяющим стоимость операции, является время передачи данных, может оказаться полезным использовать операции полусоединения. Это может дать выигрыш в скорости выполнения распределенных соединений за счет уменьшения количества данных, пересылаемых между сайтами.

Вопросы

20.1. В распределенной среде блокирующие алгоритмы могут классифицироваться как централизованные, с первичными копиями или распределенные. Дайте сравнительную оценку этих алгоритмов и укажите на существующие различия.

20.2. Один из наиболее известных методов выявления ситуаций распределенной взаимной блокировки был разработан Обермарком. Поясните принципы работы этого метода и объясните применяемые способы обнаружения и устранения взаимной блокировки.

20.3. Приведите две централизованные топологии, альтернативные по отношению к топологии двухфазной фиксации транзакций.

20.4. *Объясните смысл термина "неблокирующий протокол" и поясните, почему* двухфазный протокол фиксации транзакций нельзя считать неблокирующим.

20.5. Поясните, почему трехфазный протокол фиксации транзакций можно считать неблокирующим в случае отсутствия полного отказа сайтов.

20.6. Сравните и укажите отличия между разными схемами владения реплицируемыми данными. Проиллюстрируйте свой ответ на примерах.

20.7. Сравните различные механизмы репликации в базах данных и укажите отличия между ними.

Упражнения

20.8. Подробно опишите функционирование централизованного варианта двухфазного протокола фиксации транзакций в распределенной среде. Отдельно опишите алгоритм работы координатора и участника.

20.9. Подробно опишите функционирование трехфазного протокола фиксации транзакций в распределенной среде. Отдельно опишите алгоритм работы координатора и участника.

20.10. Проанализируйте особенности СУБД, с которой вы в настоящее время работаете, и установите предоставляемую ей степень поддержки модели X/Open DTP и репликации данных.

20.11. Исполнительный директор компании *DreamHome* предложил вам провести исследование тех требований, которые существуют в этой организации в отношении распределения данных, и подготовить отчет о потенциальной возможности использования распределенной СУБД. В отчете должно быть представлено сравнение технологий централизованной и распределенной СУБД, а также описаны преимущества и недостатки развертывания СУРБД в данной организации. Кроме того, вы должны указать потенциальные области возникновения проблем и дать оценку целесообразности использования серверов репликации как альтернативы переходу компании к работе в распределенной среде. Наконец, в отчете необходимо тщательно обосновать набор рекомендаций и предложений по выбору наиболее подходящего случаю решения.

20.12. Рассмотрим пять транзакций: T_1, T_2, T_3, T_4 и T_5 со следующими характеристиками:

T_1	инициируется на сайте S_1	и запускает агента на сайте S_2,
T_2	инициируется на сайте S_3	и запускает агента на сайте S_1,
T_3	инициируется на сайте S_1	и запускает агента на сайте S_3,
T_4	инициируется на сайте S_2	и запускает агента на сайте S_3,
T_5	инициируется на сайте S_3.	

Информация о блокировках для этих транзакций представлена в табл. 20.3.

Таблица 20.3. Информация о блокировках

Транзакция	Элемент данных, заблокированный транзакцией	Элемент данных, требуемый транзакции	Сайт, на котором выполняется операция
T_1	x_1	x_8	S_1
T_1	x_6	x_2	S_2
T_2	x_4	x_1	S_1
T_2	x_5		S_3
T_3	x_2	x_7	S_1
T_3		x_3	S_3
T_4	x_7		S_2
T_4	x_8	x_5	S_3
T_5	x_3	x_7	S_3

а) Подготовьте локальные графы ожидания для каждого сайта. Какое заключение можно сделать, исходя из вида этих графов?

б) Используйте для приведенных транзакций метод Обермарка и покажите, как с его помощью можно обнаружить ситуацию распределенной взаимной блокировки. Какое заключение можно сделать в отношении глобального графа ожидания?