1-лекц

1. Распределенные базы данных

# Введение

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

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

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

# Основные понятия и определения

2

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

**Распределенная система управления базой данных (РСУБД)** – это программная система, предназначенная для управления распределен- ными базами данных и позволяющая сделать распределенность инфор- мации прозрачной для конечного пользователя.

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

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

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

3

Поясним, что репликация заключается в поддержке актуальной ко- пии (реплики) некоторого фрагмента базы данных на нескольких различ- ных узлах.

Нет необходимости в том, чтобы на каждом из узлов системы су- ществовала своя собственная база данных, что и показано на примере то- пологии РСУБД, представленной на рис. 1.1.

Узел 1



Узел 4

База данных

Узел 2

Компьютерная сеть

База данных

База данных

Узел 3

*Рис. 1.1. Топология распределенной системы управления базой данных*

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

2-3 лекц

## Распределенная обработка

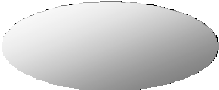
4

Рассмотрим различия, которые существуют между распределенны- ми СУБД и распределенной обработкой данных.

**Распределенная обработка** – это обработка с использованием цен- трализованной базы данных, доступ к которой может осуществляться с различных компьютеров сети.

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

Узел 1



Узел 4

Узел 2

Компьютерная сеть

База данных

Узел 3

*Рис. 1.2. Топология системы с распределенной обработкой*

Точнее, система "клиент/сервер" – это система, в которой одни уз- лы – *клиенты*, а другие – *серверы*; все данные размещены на узлах, кото- рые являются серверами; все приложения выполняются на узлах- клиентах и полная локальная независимость не предоставляется. Клиент

* это процесс, посылающий запрос на обслуживание. Архитектура клиент-сервер обеспечивает прикладным программам клиента доступ к дан- ным, которыми управляет сервер. Это основное назначение этой архитекту- ры, то есть несколько клиентов эффективно используют один сервер [3].

Возможны несколько вариантов основной схемы.

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

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

## Параллельные СУБД

Остановимся на различиях, которые существуют между распреде- ленными и параллельными СУБД.

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

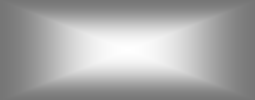
некоторых операций с целью повышения общей производительности об- работки.

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

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

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

#### ЦП ЦП ЦП ЦП



**Память Память Память Память**

**ЦП**

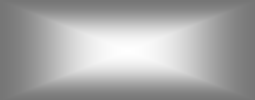
**ЦП**

**ЦП**

**ЦП**

**Сетевые соединения**

(б)



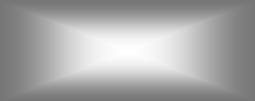
**нения**

**соеди**

**Сетевые**

(а)

**Память**



**Ц**

(в)

**Память**

**Память**

**ЦП**

**П**

**Сетевые соединения**

**ЦП**

**ЦП**

**Память**

**Память**

*Рис. 1.3. Архитектура систем с параллельной обработкой: а) с разделением памяти;*

*б) с разделением дисков; в) без разделения*

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

В состав **систем с разделением памяти** входит несколько процес- соров, разделяющих общую системную память. Эту архитектуру назы- вают также симметричной многопроцессорной обработкой. Она приме- няется для самых разных вычислительных платформ, начиная от персо- нальных ЭВМ, содержащих несколько параллельно работающих микро- процессоров, больших RISC-систем и вплоть до крупнейших мейнфрей- мов. Эта архитектура обеспечивает быстрый доступ к данным для огра- ниченного числа процессоров (обычно не больше 64). При увеличении числа процессоров сетевые взаимодействия начинают ограничивать про- изводительность всей системы.

**Системы без разделения** иначе называют системами массовой па- раллельной обработки. В таких системах каждый процессор имеет свою собственную оперативную и дисковую память. База данных распределена между всеми дисковыми устройствами вычислительных подсистем, свя- занных с этой базой данных. В результате все данные прозрачно доступ- ны пользователям каждой из вычислительных подсистем. Такая архитек- тура обеспечивает более высокую масштабируемость, чем системы с раз- делением памяти, и позволяет легко организовать поддержку работы большого количества процессоров. Однако только в случае, когда тре- буемые данные хранятся локально, удается достичь максимальной про- изводительности.

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

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

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

Вопросы

* 1. Поясните значение терминов «распределенная база данных» и

«распределенная СУБД. Назовите причины создания подобных систем.

* 1. Перечислите особенности, которые должна иметь любая РСУБД.
  2. Сравните и укажите отличия между РСУБД и системами с распре- деленной обработкой. При каких обстоятельствах выбор РСУБД оказывается предпочтительней организации распределенной обра- ботки?
  3. Сравните и укажите отличия между РСУБД и системами с парал- лельной обработкой. При каких обстоятельствах РСУБД оказыва- ется предпочтительнее параллельной СУБД?

4-лекц

# Гомогенные и гетерогенные распределенные СУБД

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

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

вые узлы к уже существующей распределенной системе. Более того, ор- ганизуя на различных узлах параллельную обработку информации, мож- но повысить производительность всей системы.

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

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

Если используется другой тип оборудования, но та же СУБД, ме- тоды выполнения трансляции заключаются в замене кодов и изменении длины слова. Если типы используемых на узлах СУБД различны, транс- ляция усложняется необходимостью отображения структуры данных од- ной модели в соответствующие структуры данных другой модели. На- пример, отношения в реляционной модели данных должны быть преоб- разованы в записи и наборы, типичные для сетевой модели данных. Кро- ме того, необходимо транслировать текст запросов с одного используе- мого языка манипулирования данными на другой. Если отличаются и тип оборудования, и тип используемой СУБД, потребуется выполнять оба вида трансляции.

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

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

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

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

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

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

# Мультибазовые системы

Одной из разновидностей распределенных СУБД являются муль- тибазовые системы.

**Мультибазовая система** – распределенная система управления ба- зами данных, в которой управление каждым из узлов осуществляется со- вершенно автономно [2].

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

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

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

Говоря простыми словами, мультибазовая СУБД является такой СУБД, которая прозрачным образом располагается поверх существую- щих баз данных и файловых систем, предоставляя их своим пользовате- лям как некоторую единую базу данных. Такая поддержка глобальной схемы позволяет пользователям на основании этой схемы строить запро- сы 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 транс- лирует глобальные запросы и превращает их в запросы и операторы мо- дификации данных соответствующих локальных СУБД. Полученные по- сле выполнения локальных запросов результаты сливаются в единый глобальный результат, предоставляемый пользователю. Кроме того, мультибазовая СУБД осуществляет контроль за выполнением фиксации или отката отдельных операций глобальных транзакций локальных СУБД, а также обеспечивает сохранение целостности данных в каждой из локальных баз данных. Программы мультибазовой СУБД управляют различными шлюзами, с помощью которых контролируют работу ло- кальных СУБД.

5-лекц

# Преимущества и недостатки распределенных СУБД

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

В распределенных системах данные размещаются на том сайте, на котором зарегистрированы пользователи, которые их чаще всего исполь- зуют. В результате пользователи этого узла получают локальный кон- троль над требуемыми им данными и могут регулировать локальные ог- раничения на их использование. В этом заключается **разделяемость и локальная автономность** распределенных СУБД.

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

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

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

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

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

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

обработки данных может препятствовать организации эффективной за- щиты от нарушений целостности данных.

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

Вопросы

* 1. В чем заключаются методы выполнения трансляции в гетероген- ных РСУБД?
  2. Каковы недостатки использования шлюзов для обеспечения про- зрачности в отношении используемой СУБД?
  3. Какие спецификации должна подготовить Specification Working Group?
  4. Сравните и укажите отличия между обычными типами РСУБД и мультибазовыми системами. При каких обстоятельствах выбор мультибазовой системы оказывается предпочтительней организа- ции обычной РСУБД?
  5. Назовите преимущества и недостатки, свойственные распределен- ным системам.

6-7лекц

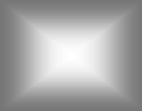
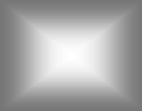
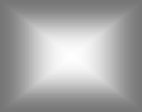
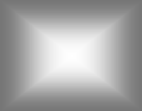
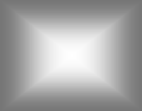
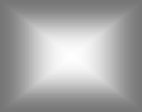
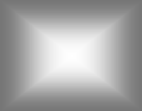
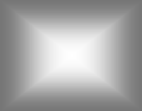
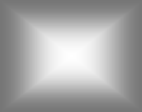
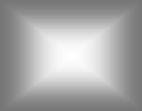
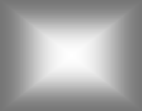
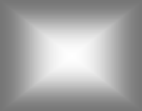
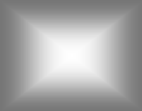
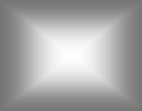
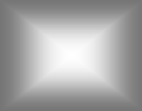
# 1.6. Архитектура распределенных СУБД

Трехуровневая архитектура ANSI-SPARC для СУБД, обсуждав- шаяся, например, в [3], представляет собой типовое решение для центра- лизованных СУБД. Однако распределенные СУБД имеют множество от- личий, которые достаточно сложно отобразить в некотором эквивалент- ном архитектурном решении, приемлемом для большинства случаев. Можно найти некоторое рекомендуемое решение, учитывающее особен- ности работы с распределенными данными. Один из примеров рекомен- дуемой архитектуры РСУБД приведен в [2] и представлен на рис. 1.4. Он включает следующие элементы:

Узел 1

Узел 2

Узел N



**Глобальная внешняя схема**

**Схема фрагмен- тации**

Узел 1

Узел 2

Узел N

**Локальная концептуальная схема**

**Локальная концептуальная схема**

**Локальная концептуальная схема**

**Локальная внутренняя схема**

**Локальная внутренняя схема**

**Локальная внутренняя схема**

**ная схе- ражения**

**Локаль ма отоб**

**ная схе- ражения**

**Локаль ма отоб**

**ная схе- ражения**

**Локаль ма отоб**

**распре- ния**

**Схема**

**деле**

**ьная уальная ма**

**Глобал концепт**

**схе**

**Глобальная внешняя схема**

**Глобальная внешняя схема**

База данных База данных База данных

*Рис. 1.4. Архитектура, рекомендуемая для РСУБД*

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

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

*Глобальная концептуальная схема* представляет собой логическое описание всей базы данных, представляющее ее так, как будто она не яв- ляется распределенной. Этот уровень РСУБД соответствует концепту- альному уровню архитектуры ANSI-SPARC и содержит определения сущностей, связей, требований защиты и ограничений поддержки цело- стности информации. Он обеспечивает физическую независимость дан- ных от распределенной среды. Логическую независимость данных обес- печивают *глобальные внешние схемы*.

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

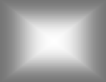
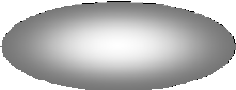
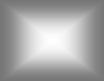
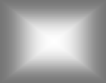
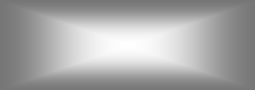
Каждая локальная СУБД имеет свой собственный набор схем. *Ло- кальная концептуальная* и *локальная внутренняя схемы* полностью соот- ветствуют уровням архитектуры ANSI-SPARC. *Локальная схема ото- бражения* используется для отображения фрагментов в схеме распреде- ления во внутренние объекты локальной базы данных. Эти элементы яв- ляются зависимыми от типа используемой СУБД и служат основой для построения гетерогенных РСУБД.

Наряду с общей архитектурой РСУБД рассмотрим детализирован- ную топологию РСУБД, которая называется *компонентной архитекту- рой* РСУБД [2] и должна включать четыре следующих важнейших ком- понента:

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

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

Узел 1



Глобальный системный каталог

**Локальная СУБД**

**Компьютерная сеть**

База данных

Глобальный системный каталог

**Компонент передачи данных**

**УБД**

**РС**

**Компонент передачи данных**

**РСУБД**

Узел 3

*Рис. 1.5. Компонентная архитектура распределенной СУБД*

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

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

*Глобальный системный каталог* выполняет те же самые функции, что и системный каталог в централизованных базах данных. Однако гло- бальный каталог содержит информацию, специфическую для распреде- ленной природы системы, например, схемы фрагментации и распределе- ния. Этот каталог сам по себе может являться распределенной базой дан- ных и поэтому может подвергаться фрагментации и распределению, быть

полностью реплицируемым или централизованным, как и любое другое отношение.

*Компонент распределенной СУБД* является управляющим по от- ношению ко всей системе элементом. Типичная РСУБД должна обеспе- чивать, по крайней мере, тот же набор функциональных возможностей, что и для централизованных СУБД. Кроме того, РСУБД должна предос- тавлять следующий набор функциональных возможностей [2].

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

Вопросы

* 1. Опишите рекомендуемую архитектуру распределенной СУБД.
  2. Опишите компонентную архитектуру распределенной СУБД.
  3. Какие функции должна выполнять распределенная СУБД?

8-лекц

# Проектирование распределенных реляционных баз данных

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

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

ли во фрагмент выделяется подмножество кортежей отношения, то он называется горизонтальным фрагментом. Фрагмент называ- ется вертикальным, если в нем используется подмножество атри- бутов отношения.

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

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

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

* + - частота запуска приложения на выполнение;
    - узел, на котором запускается приложение;
    - требования к производительности транзакций и приложений. Качественная информация может включать перечень транзакций,

выполняемых в приложении, используемые в этих транзакциях отноше- ния, атрибуты и кортежи, тип доступа к этим объектам (чтение или за- пись), предикаты, используемые в операциях чтения.

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

* + - *Локальность ссылок.* Данные должны храниться как можно бли- же к местам их использования. Если фрагмент используется не- сколькими узлами, может оказаться целесообразным разместить на этих узлах его реплики.
    - *Повышение надежности и доступности*. Надежность и доступ- ность данных повышаются за счет использования механизма реп- ликации. В случае отказа одного из узлов всегда будет существо- вать копия фрагмента, сохраняемая на другом узле.
    - *Достаточный уровень производительности*. Неправильный вы- бор схемы размещения данных может привести к возникновению

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

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

Вопросы

* 1. Одна из интенсивно развивающихся областей теории распреде- ленных систем связана с выработкой методологии разработки рас- пределенных баз данных. Назовите главные особенности, которые должны учитываться при проектировании распределенных баз данных. Поясните, как эти вопросы связаны с глобальным систем- ным каталогом.
  2. В чем состоят стратегические цели определения и распределения фрагментов?

9 лекц

## Распределение данных

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

Централизованное размещение

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

#### Локальность ссылок - самая низкая. Надежность и доступность - самая низкая. Производительность - неудовлетворительная. Стоимость устройств хранения - самая низкая.

**Затраты на передачу данных - самые высокие.**

Раздельное (фрагментированное) размещение

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

#### Локальность ссылок - высокая.

**Надежность и доступность - низкая для отдельных**

#### элементов; высокая для системы в целом.

**Производительность - удовлетворительная. Стоимость устройств хранения - самая низкая.**

#### Затраты на передачу данных - низкие.

Размещение с полной репликацией

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

#### Локальность ссылок - самая высокая. Надежность и доступность - самая высокая.

**Производительность - хорошая для операций чтения.**

#### Стоимость устройств хранения - самая высокая.

**Затраты на передачу данных - высокие для операций**

#### обновления, низкие для операций чтения.

Размещение с выборочной репликацией

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

#### Локальность ссылок - высокая.

**Надежность и доступность - низкая для отдельных**

#### элементов; высокая для системы в целом.

**Производительность - удовлетворительная. Стоимость устройств хранения - средняя.**

#### Затраты на передачу данных - низкие.

Вопросы

* 1. Сравните способы размещения данных в системе по уровню ло- кальности ссылок.
  2. Сравните способы размещения данных в системе по уровню на- дежности и доступности.
  3. Сравните способы размещения данных в системе по уровню про- изводительности.
  4. Сравните способы размещения данных в системе по уровню стои- мости устройств хранения.
  5. Сравните способы размещения данных в системе по уровню затрат на передачу данных.

10-11 лекц

## Фрагментация

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

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

Для корректности фрагментации необходимо выполнение следую- щих правил [2].

* + - 1. *Полнота*. Каждый элемент данных из исходного отношения дол- жен присутствовать, по крайней мере, в одном из созданных фрагментов. Это гарантирует отсутствие потери информации при фрагментации.
      2. *Восстановимость*. Исходное отношение должно быть восстано- вимо из его фрагментов при помощи операций реляционной ал- гебры. Это гарантирует сохранение функциональных зависимо- стей.
      3. *Непересекаемость*. Один элемент данных не должен присутство- вать в двух и более фрагментах. Исключение составляет верти- кальная фрагментация, когда в каждом фрагменте должны при- сутствовать атрибуты первичного ключа, необходимые для вос- становления исходного отношения. Это правило гарантирует ми- нимальную избыточность данных во фрагментах.

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

|  |
| --- |
|  |
|  |
|  |

|  |
| --- |
|  |
|  |
|  |

|  |
| --- |
|  |
|  |
|  |

(а)

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |

(б)

|  |  |  |
| --- | --- | --- |
|  |  |  |

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

Горизонтальная фрагментация

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

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

применяемые на одном из узлов. Если задано отношение R1, то его гори- зонтальный фрагмент может быть определен формулой [4]:

R F (R1 )

Здесь F является предикатом, построенным с использованием одно- го или больше атрибутов отношения R1.

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

*Таблица 1.1*

#### Отношение СЛУЖ

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

Пусть известно, что отделение Д1 расположено в Москве, а отделе- ния Д2 и Д3 – в Воронеже. В этом случае горизонтальная фрагментация от- ношения СЛУЖ по атрибуту Отд# может быть выполнена следующим об- разом:

М \_ СЛУЖ Отд# ' Д1'(СЛУЖ)

B \_ СЛУЖ Отд# 'Д2'Отд# 'Д3' (СЛУЖ)

В результате будут созданы два фрагмента. Первый, содержимое которого представлено в табл. 1.2, будет состоять из кортежей, в которых значение атрибута Отд# будет равно 'Д1'. Этот фрагмент имеет внутри- системное имя М\_СЛУЖ и будет храниться на узле в Москве. Второй фрагмент с именем В\_СЛУЖ (табл. 1.3) будет храниться на узле в Вороне- же и состоять из кортежей, в которых значение атрибута Отд# равно 'Д2'.

*Таблица 1.2*

#### Горизонтальный фрагмент М\_СЛУЖ отношения СЛУЖ

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

*Таблица 1.3*

#### Горизонтальный фрагмент В\_СЛУЖ отношения СЛУЖ

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

Предложенная схема фрагментации отвечает всем правилам кор- ректности.

* *Полнота*. Каждый кортеж исходного отношения присутствует либо во фрагменте М\_СЛУЖ, либо во фрагменте В\_СЛУЖ.
* *Восстановимость*. Отношение СЛУЖ может быть восстановлено из созданных фрагментов с помощью следующей операции объе- динения:

### СЛУЖ М \_ СЛУЖ  В \_ СЛУЖ

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

В одних случаях целесообразность использования горизонтальной фрагментации очевидна. Когда же это не так, потребуется выполнение детального анализа приложений. Анализ должен включать проверку пре- дикатов 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 зна- чения могут быть дискретными или представлять диапазон значений.

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

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

Вертикальная фрагментация

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

При вертикальной фрагментации в различные фрагменты объеди- няются атрибуты, используемые отдельными приложениями. Определе- ние фрагментов в этом случае выполняется с помощью операции проек- ции реляционной алгебры [4]. Для заданного отношения R1 вертикаль- ный фрагмент может быть вычислен с помощью следующей формулы:

R a1,K ,an (R1 )

Здесь a1,…,an представляют собой атрибуты отношения R1.

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

C1 Таб#,Долж,Пол,Рожд,Пасп#,Оклад (СЛУЖ)

C2 Таб#,Фио,Отд# (СЛУЖ)

С помощью этих формул будут созданы два фрагмента, содержи- мое которых представлено в табл. 1.4 и 1.5. При этом оба фрагмента со- держат первичный ключ – атрибут Таб# – что позволяет при необходи- мости реконструировать исходное отношение. Преимущество вертикаль- ной фрагментации состоит в том, что отдельные фрагменты могут раз-

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

*Таблица 1.4*

#### Вертикальный фрагмент С1 отношения СЛУЖ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Таб# | Пол | Рожд | Пасп# | Долж | Оклад |
| 154 | М | 10.11.60 | 153238 | ассистент | 3200 |
| 155 | М | 13.04.51 | 473850 | менеджер | 4000 |
| 156 | Ж | 11.05.69 | 872134 | ассистент | 2500 |
| 157 | М | 15.12.70 | 876234 | менеджер | 3000 |
| 158 | Ж | 23.03.68 | 786349 | менеджер | 3500 |
| **Вертикаль** | | *Таблица 1.5*  **ный фрагмент С2 отношения СЛУЖ** | | | |
| Таб# | Отд# | | Фио | | |
| 154 | Д1 | | Иванов И.И. | | |
| 155 | Д1 | | Петров П.П. | | |
| 156 | Д2 | | Зайцева С.П. | | |
| 157 | Д2 | | Сидоров С.С. | | |
| 158 | Д3 | | Медведева О.П. | | |

Приведенная схема фрагментации удовлетворяет правилам кор- ректности.

* *Полнота*. Каждый атрибут отношения СЛУЖ присутствует либо во фрагменте С1, либо во фрагменте С2.
* *Восстановимость*. Исходное отношение СЛУЖ может быть ре- конструировано из отдельных фрагментов с помощью операции естественного соединения:

СЛУЖ C1 >< C2

* *Непересекаемость*. Содержимое отдельных фрагментов не пере- секается, за исключением атрибута первичного ключа Таб#, необ- ходимого для реконструкции исходного отношения.

Смешанная фрагментация

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

**Смешанный** фрагмент образуется либо после дополнительной вертикальной фрагментации созданных ранее горизонтальных фрагмен- тов, либо путем горизонтальной фрагментации определенных ранее вер- тикальных фрагментов (рис 1.7).

Смешанная фрагментация определяется сочетанием операций вы- борки и проекции реляционной алгебры. Для существующего отношения R1 смешанный фрагмент можно определить по формулам:

R F (a1,K ,an (R1 ))

или

R a1,K an (F (R1 ))

Здесь F – предикат, построенный с использованием одного или бо- лее атрибутов a1,…an отношения R1.

(а) (б)

*Рис. 1.7. Смешанная фрагментация:*

*а) – горизонтально разделенные вертикальные фрагменты; б) – вертикально разделенные горизонтальные фрагменты*

***Пример 1.6.3.*** Выше отношение СЛУЖ было разбито на два верти- кальных фрагмента (табл. 1.4, 1.5) для приложений печати платежной ве- домости и некоторого документа отдела кадров. Для разбиения исполь- зовались формулы:

C1 Таб#,Долж,Пол,Рожд,Пасп#,Оклад (СЛУЖ)

C2 Таб#,Фио,Отд# (СЛУЖ)

Теперь можно выполнить дополнительную горизонтальную фрагмента- цию фрагмента С2 по атрибуту номера отделения компании Таб#:

C21 Отд#'Д1' (C2 )

C22 Отд#'Д2' (C2 ) C23 Отд#'Д3' (C2 )

В результате будут созданы три фрагмента, первый из которых включает кортежи с номером отделения, равным 'Д1' (табл. 1.7), второй – с номером отделения, равным 'Д2' (табл. 1.8), а в третий вошли кортежи с номером отделения, равным 'Д3' (табл. 1.9). Содержимое фрагмента С1 показано в табл. 1.6

*Таблица 1.6*

#### Смешанная фрагментация отношения СЛУЖ. Фрагмент С1

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

*Таблица 1.7*

#### Смешанная фрагментация отношения СЛУЖ. Фрагмент С21

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

*Таблица 1.8*

#### Смешанная фрагментация отношения СЛУЖ. Фрагмент С22

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

*Таблица 1.9*

#### Смешанная фрагментация отношения СЛУЖ. Фрагмент С23

|  |  |  |
| --- | --- | --- |
| Таб# | Отд# | Фио |
| 158 | Д3 | Медведева О.П. |

Полученная схема фрагментации удовлетворяет правилам коррект- ности.

* *Полнота*. Каждый из атрибутов исходного отношения СЛУЖ присутствует либо во фрагменте С1, либо во фрагменте С2; каждый

кортеж (в виде отдельных частей) исходного отношения присутст- вует во фрагменте С1, а также в отношении С21, С22 или С23.

* *Восстановимость*. Исходное отношение СЛУЖ может быть вос- становлено из полученных фрагментов путем выполнения опера- ций объединения и естественного соединения по следующей формуле:

СЛУЖ C1 >< (C21  C22  C23 )

* *Непересекаемость*. Полученные фрагменты не пересекаются, так как фрагменты C1 и С2 содержат различные атрибуты (за исклю- чением обязательного атрибута первичного ключа) и не сущест- вует работника, нанятого более чем в одно отделение компании.

Если отношение содержит небольшое количество кортежей, кото- рые относительно редко обновляются, то от фрагментации **отказывают- ся**. Такое отношение оставляют нефрагментированным и просто разме- щают на каждом из узлов его реплицируемые копии.

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

Вопросы

* 1. Дайте определение и укажите различия между альтернативными схемами фрагментации глобальных отношений.
  2. Поясните, как можно проверить корректность выполненных дей- ствий при фрагментации и получить гарантии того, что в процессе фрагментации в базу данных не было внесено семантических из- менений.

## 12-13 лекц

## Репликация

**Репликацию** можно определить как процесс генерации и воспро- изведения нескольких копий данных, размещаемых на одном или не- скольких узлах.

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

Виды репликации

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

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

Многие коммерческие распределенные СУБД предоставляют дру- гой механизм репликации, получивший название *асинхронной реплика- ции*. Он предусматривает обновление целевых баз данных после выпол- нения обновления исходной базы данных. Задержка в восстановлении со- гласованности данных может варьироваться от нескольких секунд до не- скольких часов или даже дней. Однако рано или поздно данные во всех копиях будут приведены в синхронное состояние. Хотя такой подход на- рушает принцип независимости распределенных данных, он вполне мо- жет пониматься как приемлемый компромисс между целостностью дан- ных и их доступностью, причем последнее может быть важнее для орга-

низаций, чья деятельность допускает работу с копией данных, необяза- тельно точно синхронизованной на текущий момент.

Функции службы репликации

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

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

Схемы владения данными

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

* «ведущий/ведомый»;
* «рабочий поток»;
* «повсеместное обновление».

Последний вариант иногда называют *одноранговой*, или *симмет- ричной репликацией*.

При организации владения данными по схеме «ведущий/ведомый» асинхронно реплицируемые данные принадлежат одному из узлов, назы- ваемому ведущим, или первичным, и могут обновляться только на нем. Здесь можно провести аналогию между издателем и подписчиками. Из- датель (ведущий узел) публикует свои данные. Все остальные узлы толь- ко 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 раз- личных, не перекрывающихся наборов данных. Однако в системе может существовать только один узел, на котором располагается ведущая об- новляемая 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 од- ной или более распределенных баз данных могут выгружаться в отдельную, локальную систему ППР, где они будут только счи- тываться при выполнении различных видов анализа.
* *Централизованное распределение или распространение инфор- мации*. Распространение данных имеет место в тех случаях, когда данные обновляются только в центральном звене системы, после чего реплицируются их копии, доступные только для чтения. Этот вариант репликации данных показан на рис.1.8, а.
* *Консолидация удаленной информации*. Консолидация данных имеет место в тех случаях, когда обновление данных выполняется локально, поле чего их копии, доступные только для чтения, от- сылаются в общее хранилище. В этой схеме каждый из сайтов ав- тономно владеет некоторой частью данных. Этот вариант репли- кации данных показан на рис. 1.8, б.

Узел



Ведущий (чтение / запись)

Узел

Ведомый (только чтение)

Ведущий (чтение / запись)

Ведомый (только чтение)

Ведущий (чтение / запись)



Узел

Узел

Узел

Ведомый (только чтение)

Ведомый (только чтение)

Ведущий (чтение / запись)

Узел

(а)

Узел

(б)

Узел

*Рис. 1.8. Владение данными по схеме «ведущий/ведомый»:*

*а) распределение данных; б)консолидация данных*

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

Ведущий сайт может владеть данными всей таблицы, и в этом слу- чае все остальные сайты являются лишь подписчиками на копии этой таблицы, доступные только для чтения. В альтернативном варианте мно- гие 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 *асим- метричной репликацией*.

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

Централизованные системы позволяют приложениям, выполняю- щим 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 вла- дения «рабочий поток» приложения могут быть распределены по различ- ным сайтам, и когда данные реплицируются и пересылаются на следую- щий сайт в цепочке, вместе с ними передается и право на их обновление, как показано на рис. 1.9.

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

**Центральный офис**

(выписка счетов)

**Отделение компании**

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

Таблица "Счета"

№

56

57

Сумма Состояние

4 987 К оплате

7 865 К оплате

Таблица "Счета"

№

56

57

Сумма Состояние

4 987 Выписан

7 865 Выписан

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

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

Сохранение целостности транзакций

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

В данном случае транзакция, состоящая на исходном сайте из не- скольких 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 тран- закций на целевом сайте завершалась успешно, а остальная часть – нет, то согласованность информации между двумя базами данных утрачива- лась.

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

Моментальные снимки таблиц

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

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

ником для получения сведений об изменениях в исходных данных. Дос- таточно 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. Различ- ные СУБД реализуют подобный механизм по-разному: в некоторых слу- чаях 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 целесооб- разно применять метод организации очередей. Если произойдет отказ се- тевого соединения или целевого сайта, сведения об изменениях могут со- храняться в очередях до тех пор, пока соединение не будет восстановле- но. 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 пользова- телям возможности создавать собственные приложения, выполняющие репликацию данных с использованием механизма *триггеров базы дан- ных*.

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

* Отслеживание запуска и выполнение триггерных процедур созда- ет 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 допол- нительную нагрузку на приложения и сетевые соединения. В про- тивоположность этому, при использовании моментальных сним- ков все выполненные изменения пересылаются за одну операцию.
* Триггеры не могут выполняться в соответствии с некоторым гра- фиком. Они выполняются в тот момент, когда происходит обнов- ление данных в ведущей таблице. Моментальные снимки могут создаваться в соответствии с установленным графиком или даже вручную. В любом случае это позволяет исключить дополнитель- ную нагрузку от репликации данных в периоды пиковой нагрузки на систему.
* Если реплицируется несколько связанных таблиц, синхронизация их репликации может быть достигнута за счет использования ме- ханизма групповых обновлений. Решить эту задачу с помощью триггеров существенно сложнее.
* Аннулирование результатов выполнения триггерной процедуры в случае отмены или отката транзакции – достаточно сложная задача.

Выявление и разрешение конфликтов

Когда несколько узлов могут независимо вносить изменения в реп- лицируемые данные, необходимо использовать некоторый механизм, по- зволяющий выявлять конфликтующие обновления данных и восстанав- ливать согласованность информации в базе. Простейший механизм обна- ружения конфликтов в отдельной таблице состоит в рассылке исходным сайтом как новых, так и исходных значений измененных данных для ка- ждой строки, которая была обновлена с момента последней синхрониза- ции копий. На целевом сайте сервер репликации должен сравнить с по- лученными значениями каждую строку в целевой базе данных, которая была локально изменена за данный период. Однако этот метод требует установки дополнительных соглашений для обнаружения других типов конфликтов, например нарушения ссылочной целостности между двумя таблицами.

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

* *Самая ранняя или самая поздняя временная отметка*. Изменяют- ся соответственно данные с самой ранней или самой поздней временной отметкой.
* *Приоритеты узлов*. Применяется обновление, поступившее с уз- ла с наибольшим приоритетом.
* *Дополняющие и усредненные обновления*. Введенные изменения обобщаются. Этот вариант разрешения конфликтов может ис- пользоваться в тех случаях, когда обновление атрибута выполня- ется операциями, записанными в форме отклонений.
* *Минимальное или максимальное значение*. Применяются обнов- ления, соответствующие столбцу с минимальным или максималь- ным значением.
* *По решению пользователя*. Администратор базы данных создает собственную процедуру разрешения конфликта. Для устранения различных типов конфликтов могут быть подготовлены различ- ные процедуры.
* *Сохранение информации для принятия решения вручную*. Сведе- ния о конфликте записываются в журнал ошибок для последую- щего анализа и устранения администратором базы данных вруч- ную.

Вопросы

* 1. В чем различия, достоинства и недостатки синхронной и асин- хронной репликации?
  2. Какие функции должна поддерживать служба репликации?
  3. Приведите примеры использования схемы владения данными «ве-

дущий/ведомый».

* 1. Опишите отличия в работе централизованной системы и распреде- ленной системы с моделью владения данными «рабочий поток» при обработке заказа.
  2. Нарушение какого свойства транзакций при репликации приводит к утрате согласованности информации между двумя базами дан- ных?
  3. В чем состоит общий подход к реализации метода моментальных снимков таблиц?
  4. Перечислите недостатки механизма триггеров базы данных.
  5. Опишите наиболее часто применяемые механизмы разрешения конфликтов.

14-15 лекц

# Обеспечение прозрачности в РСУБД

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

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

## 1.8.1. Прозрачность распределенности

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

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

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

C1 Таб#,Долж,Пол,Рожд,Пасп#,Оклад (СЛУЖ)

фрагмент расположен на сайте 1

C2 Таб#,Фио,Отд# (СЛУЖ)

C21 Отд#'Д1' (C2 ) C22 Отд#'Д2' (C2 ) C23 Отд#'Д3' (C2 )

Прозрачность фрагментации

фрагмент расположен на сайте 1 фрагмент расположен на сайте 2 фрагмент расположен на сайте 3

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

SELECT Фио

FROM СЛУЖ

WHERE Долж='менеджер';

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

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

Прозрачность расположения

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

SELECT Фио

FROM С21

WHERE Таб# IN

(SELECT Таб# FROM С1 WHERE Долж='менеджер') UNION SELECT Фио

FROM С22

WHERE Таб# IN

(SELECT Таб# FROM С1 WHERE Долж='менеджер') UNION SELECT Фио

FROM С23

WHERE Таб# IN

(SELECT Таб# FROM С1 WHERE Долж='менеджер');

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

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

Прозрачность репликации

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

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

Надо сказать, что могут существовать системы, которые не обеспе- чивают прозрачности расположения, но поддерживают прозрачность ре- пликации.

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 распо- ложение соответствующих элементов данных. Тот же самый запрос в системе с прозрачностью локального отображения приобретает следую- щий вид:

SELECT Фио

FROM С21 *AT SITE 1*

WHERE Таб# IN

(SELECT Таб# FROM С1 *AT SITE 1*

WHERE Долж='менеджер') UNION

SELECT Фио

FROM С22 *AT SITE 2*

WHERE Таб# IN

(SELECT Таб# FROM С1 *AT SITE 1*

WHERE Долж='менеджер') UNION

SELECT Фио

FROM С23 *AT SITE 3*

WHERE Таб# IN

(SELECT Таб# FROM С1 *AT SITE 1*

WHERE Долж='менеджер');

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

Прозрачность именования

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

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

Альтернативное решение состоит в использовании префиксов, по- мещаемых в имена объектов в качестве идентификатора узла, создавшего этот объект [2]. Например, отношение ДЕТАЛИ, созданное на узле У1, могло бы получить имя Уl.ДЕТАЛИ. Аналогичным образом, необходимо иметь возможность идентифицировать каждый фрагмент и каждую его копию. Поэтому второй копии третьего фрагмента отношения ДЕТАЛИ, созданного на узле У1, можно было бы присвоить имя Уl.ДЕТАЛИ.Ф3.К2. Однако подобный подход приводит к утрате прозрачности распределенно- сти.

Подход, который позволяет преодолеть недостатки, свойственные обоим упомянутым методам, состоит в использовании **алиасов** (вымыш- ленных имен, псевдонимов), создаваемых для каждого из объектов базы данных. В результате объект Уl.ДЕТАЛИ.Ф3.К2 пользователям узла У1 может быть известен под именем ДЕТАЛИ\_ЛОКАЛЬНЫЙ. Задача пре- образования алиаса в истинное имя соответствующего объекта базы дан- ных возлагается на РСУБД.

Вопросы

* 1. Какие уровни прозрачности должны поддерживаться РСУБД? Обоснуйте ваш ответ.
  2. РСУБД должна гарантировать, что на любой паре узлов невоз- можно будет создать объект базы данных с одним и тем же име- нем. Одно из решений этой проблемы состоит в организации сер- вера имен. Какие недостатки свойственны этому подходу? Пред- ложите альтернативный подход, свободный от указанных недос- татков.