ВВЕДЕНИЕ

Современные объемы хранимых данных, обязательные требования к их доступности и скорости обработки, динамика развития систем обуславливают важность исследования факторов, влияющих на качество баз данных, лежащих в основе современных информационных систем.

К концу 80-х годов возникли новые условия работы для БД: большие объемы информации возникают во многих местах (например, розничная торговля, полиграфическое и другие производства). Источником большого количества данных мог быть и центр, но к этим данным требуется быстрый доступ с периферии (географически распределенное производство, работающее по одному графику). К тому же данные могут запрашиваться и центром, и удаленными потребителями в удаленных местах. Имеется большое количество данных, которые используются в срочных запросах, чаще всего местного характера (продажа авиа- и железнодорожных билетов).

Во многих производствах, например, в компьютерном интегрированном полиграфическом производстве, необходимыми являются распределенные базы данных, связывающие в единое целое процесс управления комплексом различных технологических процессов. Здесь осуществляется работа не с одним приложением, а с системой приложений.

Централизованные БД, особенно построенные на классическом подходе, не могли удовлетворить новым требованиям.

Быстрое распространение сетей передачи данных, резкое увеличение объема внешней памяти ПК при ее удешевлении в 80-е годы способствовали широкому внедрению распределенных баз данных.

К достоинствам распределенных баз данных относятся:

1) соответствие структуры распределенных баз данных структуре организаций;

2) гибкое взаимодействие локальных БД;

3) широкие возможности централизации узлов;

4) непосредственный доступ к информации, снижение стоимости передач (за счет уплотнения и концентрации данных);

5) высокие системные характеристики (малое время отклика за счет распараллеливания процессов, высокая надежность);

6) модульная реализация взаимодействия, расширения аппаратных средств, возможность использования объектно-ориентированного подхода в программировании;

7) возможность распределения файлов в соответствии с их активностью;

8) независимые разработки локальных БД через стандартный интерфейс.

Вместе с тем распределенные базы данных обладают более сложной структурой, что вызывает появление дополнительных проблем (избыточность, несогласованность данных по времени, согласование процессов обновления и запросов, использования телекоммуникационных ресурсов, учет работы дополнительно подсоединенных локальных БД, стандартизация общего интерфейса) согласования работы элементов.

Серьезные проблемы возникают при интеграции в рамках распределенных баз данных однородных (гомогенных) локальных БД с одинаковыми, чаще всего реляционными, моделями данных.

Проблемы значительно усложняются, если локальные БД построены с использованием различных моделей данных (неоднородные, гетерогенные распределенные базы данных).

Целью данной курсовой работы является исследование распределенных баз данных и распределенных СУБД. Для достижения поставленной цели в работе были реализованы следующие задачи:

* Рассмотрено понятие распределенных баз данных;
* Рассмотрены свойства распределенных БД;
* Рассмотрено понятие целостности данных;
* Рассмотрен принцип построения распределенных баз данных на примере SYSTEM R\*
* Разработано приложение в среде Delphi.

1 РАСПРЕДЕЛЕННЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ И БАЗЫ ДАННЫХ

* 1. **Понятие распределенных баз данных**

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

Основная задача систем управления распределенными базами данных состоит в обеспечении средства интеграции локальных баз данных, располагающихся в некоторых узлах вычислительной сети, с тем, чтобы пользователь, работающий в любом узле сети, имел доступ ко всем этим базам данных как к единой базе данных [1].

При этом должны обеспечиваться:

* простота использования системы;
* возможности автономного функционирования при нарушениях связности сети или при административных потребностях;
* высокая степень эффективности.

Возможны однородные и неоднородные распределенные базы данных. В однородном случае каждая локальная база данных управляется одной и той же СУБД. В неоднородной системе локальные базы данных могут относиться даже к разным моделям данных. Сетевая интеграция неоднородных баз данных - это актуальная, но очень сложная проблема. Многие решения известны на теоретическом уровне, но пока не удается справиться с главной проблемой - недостаточной эффективностью интегрированных систем. Заметим, что более успешно практически решается промежуточная задача - интеграция неоднородных SQL-ориентированных систем. Понятно, что этому в большой степени способствует стандартизация языка SQL и общее следование производителей СУБД принципам открытых систем [2].

**1.2 Свойства распределенных баз данных**

Определение распределенных баз данных (DDB) предложил Дэйт (C.J. Date). Он установил 12 свойств или качеств идеальной DDB [4]:

* Локальная автономия (local autonomy)
* Независимость узлов (no reliance on central site)
* Непрерывные операции (continuous operation)
* Прозрачность расположения (location independence)
* Прозрачная фрагментация (fragmentation independence)
* Прозрачное тиражирование (replication independence)
* Обработка распределенных запросов (distributed query processing)
* Обработка распределенных транзакций (distributed transaction processing)
* Независимость от оборудования (hardware independence)
* Независимость от операционных систем (operationg system independence)
* Прозрачность сети (network independence)
* Независимость от баз данных (database independence)

Локальная автономия. Это качество означает, что управление данными на каждом из узлов распределенной системы выполняется локально. Будучи фрагментом общего пространства данных, БД , в то же время функционирует как полноценная локальная база данных; управление ею выполняется локально и независимо от других узлов системы [4].

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

Непрерывные операции. Это качество можно трактовать как возможность непрерывного доступа к данным (известное «24 часа в сутки, семь дней в неделю») в рамках DDB вне зависимости от их расположения и вне зависимости от операций, выполняемых на локальных узлах. Это качество можно выразить лозунгом «данные доступны всегда, а операции над ними выполняются непрерывно» [4].

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

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

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

SELECT \* FROM employee@phoenix, employee@denver ORDER BY emp\_id

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

SELECT employee.emp\_id, emp\_name, salary FROM employee@denver, employee@phoenix, emp\_salary@dallas ORDER BY emp\_id

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

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

SELECT customer.name, customer.address, order.number, order.date FROM customer@london, order@paris WHERE customer.cust\_number = order.cust\_number

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

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

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

Прозрачность сети. Доступ к любым базам данных осуществляется по сети. Спектр поддерживаемых конкретной СУБД сетевых протоколов не должен быть ограничением системы с распределенными базами данных. Данное качество формулируется максимально широко - в распределенной системе возможны любые сетевые протоколы.

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

Исходя из определения Дэйта, можно рассматривать DDB как слабосвязанную сетевую структуру, узлы которой представляют собой локальные базы данных. Локальные базы данных автономны, независимы и самоопределены; доступ к ним обеспечиваются СУБД, в общем случае от различных поставщиков. Связи между узлами - это потоки тиражируемых данных. Топология DDB варьируется в широком диапазоне - возможны варианты иерархии, структур типа «звезда» и т.д. В целом топология DDB определяется географией информационной системы и направленностью потоков тиражирования данных [4].

**1.3 Целостность данных**

В DDB поддержка целостности и согласованности данных, ввиду свойств 1-2, представляет собой сложную проблему. Ее решение - синхронное и согласованное изменение данных в нескольких локальных базах данных, составляющих DDB - достигается применением протокола двухфазной фиксации транзакций. Если DDB однородна - то есть на всех узлах данные хранятся в формате одной базы и на всех узлах функционирует одна и та же СУБД, то используется механизм двухфазной фиксации транзакций данной СУБД. В случае же неоднородности DDB для обеспечения согласованных изменений в нескольких базах данных используют менеджеры распределенных транзакций. Это, однако, возможно, если участники обработки распределенной транзакции - СУБД, функционирующие на узлах системы, поддерживают XA-интерфейс, определенный в спецификации DTP консорциума X/Open. В настоящее время XA-интерфейс имеют СУБД CA-OpenIngres, Informix, Microsoft SQL Server, Oracle, Sybase.

Если в DDB предусмотрено тиражирование данных, то это сразу предъявляет дополнительные жесткие требования к дисциплине поддержки целостности данных на узлах, куда направлены потоки тиражируемых данных. Проблема в том, что изменения в данных инициируются как локально - на данном узле - так и извне, посредством тиражирования. Неизбежно возникают конфликты по изменениям, которые необходимо отслеживать и разрешать [4].

Обработка распределенных запросов. Обработка распределенных запросов (Distributed Query -DQ) - задача, более сложная, нежели обработка локальных и она требует решения с помощью особого компонента - оптимизатора DQ. Обратимся к базе данных, распределенной по двум узлам сети. Таблица detail хранится на одном узле, таблица supplier - на другом. Размер первой таблицы - 10000 строк, размер второй - 100 строк (множество деталей поставляется небольшим числом поставщиков). Результирующая таблица представляет собой объединение таблиц detail и supplier. Данный запрос - распределенный, так как затрагивает таблицы, принадлежащие различным локальным базам данных. Для его нормального выполнения необходимо иметь обе исходные таблицы на одном узле. Следовательно, одна из таблиц должна быть передана по сети. Очевидно, что это должна быть таблица меньшего размера, то есть таблица supplier. Следовательно, оптимизатор DQ запросов должен учитывать такие параметры, как, в первую очередь, размер таблиц, статистику распределения данных по узлам, объем данных, передаваемых между узлами, скорость коммуникационных линий, структуры хранения данных, соотношение производительности процессоров на разных узлах и т.д. От интеллекта оптимизатора DQ впрямую зависит скорость выполнения распределенных запросов.

Межоперабельность. В контексте DDB межоперабельность означает две вещи.

Во-первых, - это качество, позволяющее обмениваться данными между базами данных различных поставщиков.

Во-вторых, это возможность некоторого унифицированного доступа к данным в DDB из приложения. Возможны как универсальные решения (стандарт ODBC), так и специализированные подходы. Очевидный недостаток ODBC - недоступность для приложения многих механизмов каждой конкретной СУБД, поскольку они могут быть использованы в большинстве случаев только через расширения SQL в диалекте языка данной СУБД, но в стандарте ODBC эти расширения не поддерживаются.   
Специальные подходы - это, например, использование шлюзов, позволяющее приложениям оперировать над базами данных в "чужом" формате так, как будто это собственные базы данных. Вообще, цель шлюза - организация доступа к унаследованным (legacy) базам данных и служит для решения задач согласования форматов баз данных при переходе к какой-либо одной СУБД. Следовательно, шлюзы можно рассматривать как средство, облегчающее миграцию, но не как универсальное средство межоперабельности в распределенной системе. Вообще, универсального рецепта решения задачи межоперабельности в этом контексте не существует - все определяется конкретной ситуацией, историей информационной системы и массой других факторов [2].

Технология тиражирования данных. Тиражирование данных (Data Replication - DR) - это асинхронный перенос изменений объектов исходной базы данных в базы, принадлежащие различным узлам распределенной системы. Функции DR выполняет, как правило, специальный модуль СУБД - сервер тиражирования данных, называемый репликатором (так устроены СУБД CA-OpenIngres и Sybase). В Informix-OnLine Dynamic Server репликатор встроен в сервер, в Oracle 7 для использования DR необходимо приобрести дополнительно к Oracle7 DBMS опцию Replication Option [7].

Детали тиражирования данных полностью скрыты от прикладной программы. В этом, собственно, состоит качество 6 в определении Дэйта. Синхронное обновление DDB и DR-технология - в определенном смысле антиподы. Краеугольный камень первой - синхронное завершение транзакций одновременно на нескольких узлах распределенной системы. Ee "Ахиллесова пята" - жесткие требования к производительности и надежности каналов связи. Если база данных распределена по нескольким территориально удаленным узлам, объединенным медленными и ненадежными каналами связи, а число одновременно работающих пользователей составляет сотни и выше, то вероятность того, что распределенная транзакция будет зафиксирована в обозримом временном интервале, становится чрезвычайно малой. В таких условиях (характерных, кстати, для большинства отечественных организаций) обработка распределенных данных практически невозможна.  
DR-технология не требует синхронной фиксации изменений, и в этом ее сильная сторона. Можно накапливать изменения в данных в виде транзакций в одном узле и периодически копировать эти изменения на другие узлы.

Преимущества DR-технологии. Во-первых, данные всегда расположены там, где они обрабатываются - следовательно, скорость доступа к ним существенно увеличивается. Во-вторых, передача только операций, изменяющих данные (а не всех операций доступа к удаленным данным позволяет значительно уменьшить трафик. В-третьих, со стороны исходной базы для принимающих баз репликатор выступает как процесс, инициированный одним пользователем, в то время как в физически распределенной среде с каждым локальным сервером работают все пользователи распределенной системы, конкурирующие за ресурсы друг с другом. Наконец, в-четвертых, никакой продолжительный сбой связи не в состоянии нарушить передачу изменений. Дело в том, что тиражирование предполагает буферизацию потока изменений (транзакций); после восстановления связи передача возобновляется с той транзакции, на которой тиражирование было прервано [7].

DR-технология данных не лишена недостатков. Например, невозможно полностью исключить конфликты между двумя версиями одной и той же записи. Он может возникнуть, когда вследствие асинхронности два пользователя на разных узлах исправят одну и ту же запись в тот момент, пока изменения в данных из первой базы данных еще не были перенесены во вторую. При проектировании распределенной среды с использованием DR-технологии необходимо предусмотреть конфликтные ситуации и запрограммировать репликатор на какой-либо вариант их разрешения. В этом смысле применение DR-технологии - наиболее сильная угроза целостности DDB.

Прозрачность расположения. Это качество DDB в реальных продуктах должно поддерживаться соответствующими механизмами. Разработчики СУБД придерживаются различных подходов. Рассмотрим пример из Oracle. Допустим, что DDB включает локальную базу данных, которая размещена на узле в Лондоне. Создадим вначале ссылку (database link), связав ее с символическим именем (london\_unix), транслируемым в IP-адрес узла в Лондоне.

CREATE PUBLIC DATABASE LINK london.com CONNECT TO london\_unix USING oracle\_user\_ID;

Теперь мы можем явно обращаться к базе данных на этом узле, запрашивая, например, в операторе SELECT таблицу, хранящуюся в этой базе:

SELECT customer.cust\_name, order.order\_date FROM customer@london.com, order WHERE customer.cust\_number = order.cust\_number;

Очевидно, однако, что мы написали запрос, зависящий от расположения базы данных, поскольку явно использовали в нем ссылку. Определим customer и customer@london.com как синонимы:

CREATE SYNONYM customer FOR customer@london.com;

и в результате можем написать полностью независимый от расположения базы данных запрос:

SELECT customer.cust\_name, order.order\_date FROM customer, order WHERE customer.cust\_number = order.cust\_number

Задача решается с помощью оператора SQL CREATE SYNONYM, который позволяет создавать новые имена для существующих таблиц. При этом оказывается возможным обращаться к другим базам данных и к другим компьютерам.

Во многих СУБД задача управления именами объектов DDB решается путем использования глобального словаря данных, хранящего информацию о DDB: расположение данных, возможности других СУБД (если используются шлюзы), сведения о скорости передачи по сети с различной топологией и т.д.

**1.4 Распределенная система управления базами данных System R\***

**1.4.1 Основная цель проекта System R**

Мы ограничимся рассмотрением проблем однородных распределенных СУБД на примере System R\*.

Основную цель проекта можно сформулировать следующим образом: обеспечить средства интеграции локальных баз данных System R, располагающихся в узлах вычислительной сети, с тем, чтобы пользователь, работающий в любом узле сети, имел доступ ко всем этим базам данных так, как если бы они были централизованы. При этом должны обеспечиваться:

* легкость использования системы;
* возможности автономного функционирования при нарушениях связности сети или при административных потребностях;
* высокая степень эффективности.

Для решения этих проблем было необходимо принять ряд проектных решений, касающихся декомпозиции исходного запроса, оптимального выбора способа выполнения запроса, согласованного выполнения транзакций, обеспечения синхронизации, обнаружения и разрешения распределенных тупиков, восстановления состояния баз данных после разного рода сбоев узлов сети.

Легкость использования системы достигается за счет того, что пользователи System R\* (разработчики прикладных программ и конечные пользователи) остаются в среде языка SQL, т.е. могут продолжать работать в тех же внешних условиях, что и в System R (и SQL/DS и DB2). Возможность использования SQL основывается на обеспечении System R\* прозрачности местоположения данных. Система автоматически обнаруживает текущее местоположение упоминаемых в запросе пользователя объектов данных; одна и та же прикладная программа, включающая предложения SQL, может быть выполнена в разных узлах сети. При этом в каждом узле сети на этапе компиляции запроса выбирается наиболее оптимальный план выполнения запроса в соответствии с расположением данных в распределенной системе.

Обеспечению автономности узлов сети в System R\* уделяется очень большое внимание. Каждая локальная база данных администрируется независимо от других. Возможны автономное подключение новых пользователей, смена версии автономной части системы и т.д. Система спроектирована таким образом, что в ней не требуются централизованные службы именования объектов или обнаружения тупиков. В индивидуальных узлах не требуется наличие глобального знания об операциях, выполняющихся в других узлах сети; работа с доступными базами данных может продолжаться при выходе из строя отдельных узлов сети или линий связи [7].

**1.4.2 Средства повышения эффективности**

Высокая степень эффективности системы является одним из наиболее ключевых требований к распределенным системам управления базами данных вообще и к System R\* в частности. Для достижения этой цели используются два основных приема.

Во-первых, как и в System R, в System R\* выполнению запроса предшествует его компиляция. В ходе этого процесса производится поиск употребляемых в запросе имен объектов баз данных в распределенном каталоге и замена имен на внутренние идентификаторы; проверка прав доступа пользователя, от имени которого производится компиляция, на выполнение соответствующих операций над базами данных и выбор наиболее оптимального глобального плана выполнения запроса, который затем подвергается декомпозиции и по частям рассылается в соответствующие узлы сети, где производится выбор оптимальных локальных планов выполнения компонентов запроса и происходит генерация модулей доступа в машинных кодах. В результате множество действий производится на стадии компиляции до реального выполнения запроса. Обработанная посредством прекомпилятора System R\* прикладная программа, включающая предложения SQL, может в дальнейшем выполняться много раз без дополнительных накладных расходов. Использование распределенного каталога, распределенная компиляция и оптимизация запросов являются наиболее интересными и оригинальными аспектами проекта System R\*.

Вторым средством повышения эффективности системы является возможность перемещения удаленных отношений в локальную базу данных. Диалект SQL, используемый в System R\*, включает предложение MIGRATE TABLE, при выполнении которого указанное отношение переносится в локальную базу данных. Это средство, находящееся в распоряжении пользователей, конечно, в ряде случаев может помочь добиться более эффективного прохождения транзакций. Естественно, как и для всех операций, операция MIGRATE по отношению к указанному отношению доступна не любому пользователю, а лишь тем, которые обладают соответствующим правом.

Прежде, чем перейти к более детальному изложению наиболее интересных аспектов реализации System R\*, упомянем некоторые средства, которые разработчики этой системы предполагали реализовать на начальной стадии проекта, но которые реализованы не были (причем некоторые из них, видимо, и не будут никогда реализованы). Предполагалось иметь в системе средства горизонтального и вертикального разделения отношений распределенной базы данных, средства дублирования отношений в нескольких узлах с поддержкой согласованности копий и средства поддержания мгновенных снимков состояния баз данных в соответствии с заданным запросом.

Для задания горизонтального разделения отношений в SQL была введена конструкция вида

DISTRIBUTE TABLE <table-name> HORIZONTALLY INTO

<name> WHERE <predicate> IN SEGMENT <segment-name site>

<name> WHERE <predicate> IN SEGMENT <segment-name site>

При выполнении предложения такого типа указанное отношение разбивалось на ряд подотношений, содержащих кортежи, удовлетворяющие соответствующему предикату из раздела WHERE, и каждое полученное таким образом подотношение посылалось в казанный узел для хранения в сегменте с указанным именем. Гарантируется согласованное состояние разделов при изменении отношения.

Вертикальное разделение производилось с помощью оператора

DISTRIBUTE TABLE <table-name> VERTICALLY INTO

<name> WHERE <column-name-list> IN SEGMENT <segment-name site>

<name> WHERE <column-name-list> IN SEGMENT <segment-name site>

При выполнении такого предложения также образовывался набор подотношений с помощью проекции заданного отношения на атрибуты из заданного списка. Каждое полученное подотношение затем посылалось для хранения в сегменте с указанным именем в соответствующий узел. После этого система ответственна за поддержание согласованного состояния образованных разделов [7].

Горизонтальное и вертикальное разделение отношений реально не используются в System R\*, хотя очевидно, что выполнение собственно оператора DISTRIBUTE никаких технических трудностей не вызывает. Трудности возникают при обеспечении согласованности разделов (смотри ниже). Кроме того, разделенные отношения очень трудно использовать. В соответствии с идеологией системы учет наличия разделов отношения в разных узлах сети должен производить оптимизатор, т.е. количество потенциально возможных планов выполнения запросов, которые должны оцениваться оптимизатором, еще более возрастает. При том, что в распределенной системе число возможных планов и так очень велико, и оптимизатор работает на пределе сложности, разумным образом использовать разделенные отношения невозможно. Разработчики оптимизатора System R\* не были в состоянии учитывать разделенность отношений. Поэтому и вводить в систему разделенные отношения пока бессмысленно.

Для задания требования поддержки копий отношения в нескольких узлах сети предлагалось использовать новую конструкцию SQL

DISTRIBUTE TABLE <table-name> REPLICATED INTO

<name> IN SEGMENT <segment-name site>

<name> IN SEGMENT <segment-name site>

При выполнении такого предложения должна была производиться рассылка копий указанного отношения для хранения в именованных сегментах указанных узлов сети. Система должна автоматически поддерживать согласованность копий.

Как и в случае разделенных отношений, кроме существенных проблем поддержания согласованности копий, проблемой является и разумное использование копий, наличие которых должно было бы учитываться оптимизатором.

Создание мгновенного снимка состояния баз данных в соответствии с заданным запросом на выборку должно было производиться с использованием новой конструкции SQL.

DEFINE SNAPSHOT <snapshot-name> (<attribute-list>)

AS <query>

REFRESHED EVERY <period>

При выполнении предложения фактически производится выполнение указанного в нем запроса на выборку, а результирующее отношение сохраняется под указанным в предложении именем в локальной базе данных в том узле, в котором выполняется предложение. После этого мгновенный снимок периодически обновляется в соответствии с запомненным запросом.

Можно обновить мгновенный снимок, не дожидаясь истечения временного интервала, указанного в определении, путем выполнения предложения REFRESH SNAPSHOT <snapshot-name>.

Разумное использование мгновенных снимков более реально, чем использование разделенных отношений и копированных отношений, поскольку их можно в некотором смысле рассматривать как материализованные представления базы данных. Имя мгновенного снимка можно было бы использовать прямо в запросе на выборку там, где можно использовать имена базовых отношений или представлений. Большие проблемы связаны с обновлением отношений через их мгновенные снимки, поскольку в момент обновления содержимое мгновенного снимка может расходиться с текущим содержимым базового отношения [7].

По отношению к мгновенным снимкам проблем поддержания согласованного состояния мгновенного снимка и базовых отношений не существует, поскольку автоматическое согласование не требуется. Что же касается разделенных отношений и раскопированных отношений, то для них эта проблема общая и достаточно трудная. Во-первых, согласование разделов и копий вызывает существенные накладные расходы при выполнении операций модификации хранимых отношений. Для этого требуется выработка и соблюдение специальных протоколов модификации.

Во-вторых, введение копированных отношений обычно производится не столько для увеличения эффективности системы, сколько для увеличения доступности данных при нарушении связности сети. В системах, в которых применяется этот подход, при нарушении связности сети работа с распределенной базой данных обычно продолжается только в одной из образовавшихся подсетей. При этом для выбора подсети используются алгоритмы голосования; решение принимается на основе учета количества связных узлов сети. Применяются и другие подходы, но все они очень дорогостоящие, а самое главное, они плохо согласуются с базовым подходом System R\* по поводу выбора способа выполнения запроса на стадии его компиляции. Поэтому, как нам кажется, в System R\* никогда не будут реализованы средства, позволяющие тем или иным способом поддерживать копии отношений в нескольких узлах сети.

Далее мы рассмотрим аспекты проекта System R\*, которые нашли отражение в ее реализации и являются на наш взгляд наиболее интересными: средства именования объектов и организацию распределенного каталога баз данных; подход к распределенным компиляции и выполнению запросов; особенности использования представлений; средства оптимизации запросов; особенности управления транзакциями; средства синхронизации и распределенный алгоритм обнаружения синхронизационных тупиков.

###### 1.4.3 Именование объектов и организация распределенного каталога

Напомним прежде всего, что полное имя отношения (базового или представления) в базе данных System R имеет вид имя-пользователя.имя-отношения, где имя-пользователя идентифицирует пользователя - создателя отношения, а имя-отношения - это то имя, которое было указано в предложениях CREATE TABLE или CREATE VIEW. В запросах можно указывать либо это полное имя отношения, либо его локальное имя. Во втором случае при компиляции используются стандартные правила дополнения локального имени до полного с использованием в качестве составляющей имя-пользователя идентификатора пользователя, от имени которого выполняется компиляция.

В System R\* используется развитие этого подхода. Системное имя отношения включает четыре компонента: идентификатор пользователя-создателя отношения; идентификатор узла сети, в котором выполнялась операция создания отношения; локальное имя отношения, присвоенное ему при создании; идентификатор узла, в котором отношение располагалось непосредственно после своего создания (напомним, что отношение может перемещаться из одного узла в другой при выполнении операции MIGRATE).

В запросе на SQL можно использовать системные имена объектов, но разрешается использовать и короткие локальные имена (либо локальное имя, квалифицированное именем пользователя). При этом возможны две интерпретации локального имени. Оно может интерпретироваться как часть системного имени, и в этом случае по умолчанию дополняется до системного, исходя из идентификатора узла, в котором производится компиляция, и идентификатора пользователя, от имени которого она производится (если имя пользователя не указано явно). Вторая возможная интерпретация локального имени заключается в рассмотрении его как имени ранее определенного синонима системного имени.

Для определения синонимов SQL расширен оператором вида:

DEFINE SYNONYM <relation-name> AS <system-wide-name>.

При выполнении такого предложения в локальный каталог заносится соответствующая информация.

Таким образом, при компиляции запроса всегда можно определить системные имена всех употребляемых в нем отношений: либо они явно указаны, либо могут быть получены на основе информации из локальных отношений-каталогов.

Концепция распределенного каталога System R\* основана на наличии у каждого объекта распределенной базы данных уникального системного имени. Принято следующее соглашение: информация о размещении любого объекта базы данных (идентификатор текущего узла, в котором размещен объект) сохраняется в локальном каталоге того узла, в котором объект располагался непосредственно после создания (родового узла) [7].

Следовательно, для получения полной информации об отношении в общем случае необходимо сначала воспользоваться локальным каталогом узла, в котором происходит компиляция, затем обратиться к удаленному каталогу родового узла данного отношения и в заключение воспользоваться каталогом текущего узла. Таким образом, для получения точной системной информации о любом отношении распределенной базы данных может потребоваться самое большее два удаленных доступа к отношениям-каталогам.

Применяется некоторая оптимизация этой процедуры. В локальном каталоге узла могут храниться копии элементов каталога других узлов (своего рода кэш-каталог). Согласованность копий элементов каталога не поддерживается. Эта информация используется на первой стадии компиляции запроса (мы рассматриваем распределенную компиляцию в следующем подразделе), а затем, на второй стадии, если информация, касающаяся некоторого объекта, оказалась неточной, она уточняется на основе локального каталога того узла, в котором объект хранится в настоящее время. Обнаружение некорректности копии элемента каталога производится за счет наличия при каждом элементе каталога номера версии. Если учесть достаточную инерционность системной информации, эта оптимизация может оказаться существенной.