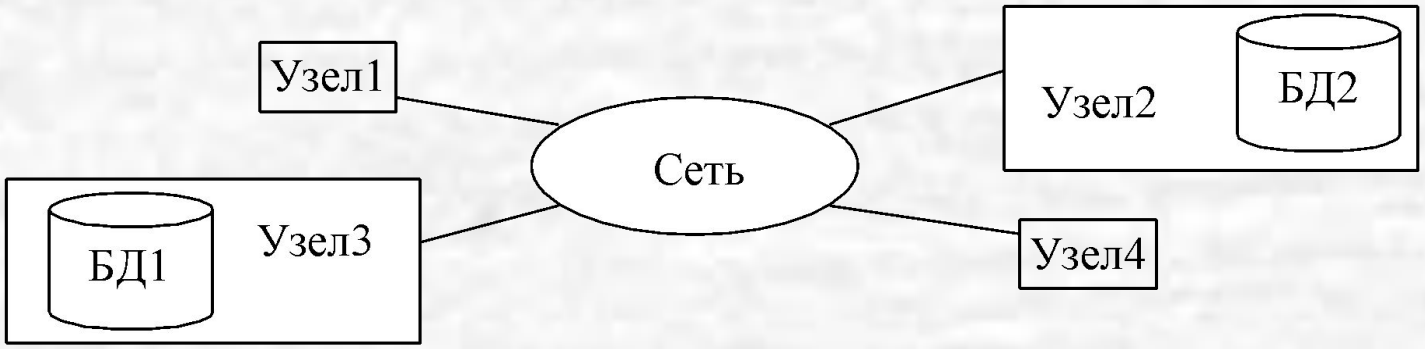
Распределенные базы данных

Общие принципы

Под **распределенной базой данных (РБД)** понимается набор логически связанных между собой разделяемых данных, которые физически распределены по разных узлам компьютерной сети.
СУРБД – это программный комплекс (СУБД), предназначенный для управления РБД и позволяющий сделать распределенность прозрачной для конечного пользователя. **Прозрачность РБД** заключается в том, что с точки зрения конечного пользователя она должна вести себя точно также, как централизованная.
Логически единая БД разделяется на фрагменты, каждый из которых хранится на одном компьютере, а все компьютеры соединены линиями связи. Каждый из этих фрагментов работает под управлением своей СУБД.



Критерии распределенности (по К. Дейту)

Локальная автономность. Локальные данные принадлежат локальным узлам и управляется администраторами локальных БД.

Локальные процессы в РБД остаются локальными.

Все процессы на локальном узле контролируются только этим узлом.

Отсутствие опоры на центральный узел.

В системе не должно быть узла, без которого система не может функционировать, т.е. не должно быть центральных служб.

Непрерывное функционирование.

Удаление или добавление узла не должно требовать остановки системы в целом.

Независимость от местоположения.

Пользователь должен получать доступ к любым данным в системе, независимо от того, являются эти данные локальными или удалёнными.

Независимость от фрагментации.

Доступ к данным не должен зависеть от наличия или отсутствия фрагментации и от типа фрагментации.

Независимость от репликации.

Доступ к данным не должен зависеть от наличия или отсутствия реплик данных.

Критерии распределенности (по К. Дейту)

Обработка распределенных запросов.

Система должна автоматически определять методы выполнения соединения (объединения) данных.

Обработка распределенных транзакций.

Протокол обработки распределённой транзакции должен обеспечивать выполнение четырёх основных свойств транзакции: атомарность, согласованность, изолированность и продолжительность.

Независимость от типа оборудования.

СУРБД должна функционировать на оборудовании с различными вычислительными платформами.

Независимость от операционной системы.

СУРБД должна функционировать под управлением различных ОС.

Независимость от сетевой архитектуры.

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

Независимость от типа СУБД.

СУРБД должна быть способной функционировать поверх различных локальных СУБД, возможно, с различными моделями данных (требование гетерогенности).

Методы поддержки распределенных данных

Существуют различные методы поддержки распределенности:

1. **Фрагментация** – разбиение БД или таблицы на несколько частей и хранение этих частей на разных узлах РБД.
2. **Репликация** – создание и хранение копий одних и тех же данных на разных узлах РБД.
3. **Распределенные ограничения целостности** – ограничения, для проверки выполнения которых требуется обращение к другому узлу РБД.
4. **Распределенные запросы** – это запросы на чтение, обращающиеся более чем к одному узлу РБД.
5. **Распределенные транзакции** – команды на изменение данных, обращающиеся более чем к одному узлу РБД.

Фрагментация

Фрагментация – основной способ организации РБД.
Назначение: хранение данных на том узле, где они чаще используются.
Основные проблемы, которые при этом возникают:
– прозрачность написания запросов к данным;
– поддержка распределенных ограничений целостности.

Схема фрагментации отношения должна удовлетворять трем условиям:

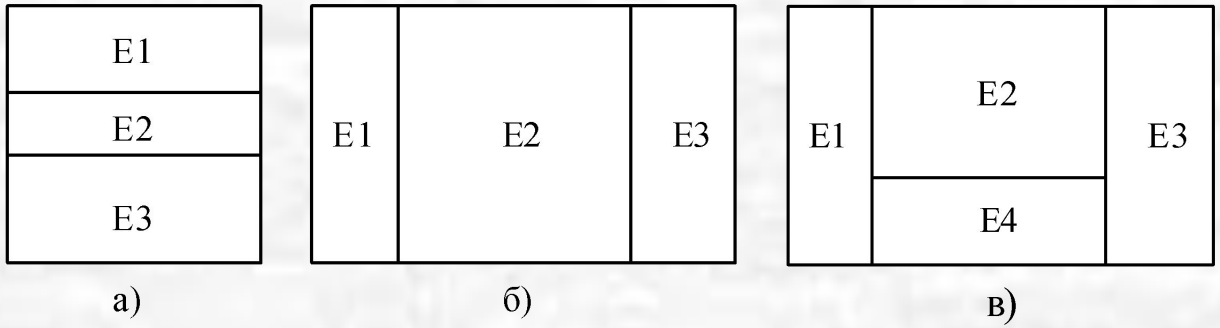
Полнота: если отношение R разбивается на фрагменты R1, R2,…, Rn, то
$U Ri = R$
(Каждый кортеж должен входить хотя бы в один фрагмент).

Восстановимость: должна существовать операция реляционной алгебры, позволяющая восстановить отношение R из его фрагментов. Это правило гарантирует сохранение функциональных зависимостей.

Непересекаемость: если элемент данных $dj \subset Ri$, то он не должен присутствовать одновременно в других фрагментах. Исключение составляет первичный ключ при вертикальной фрагментации. Это правило гарантирует минимальную избыточность данных.

Фрагментация

Типы фрагментации:
а) горизонтальная;
б) вертикальная;
в) смешанная;
г) производная.



а) б) в)

Производная фрагментация строится для подчиненного отношения на основе фрагментов родительского отношения. Например, для фрагментов отношения Emp (сотрудники) Ei подчиненное отношение "Дети" (Child), информацию о которых также целесообразно хранить в соответствующих узлах, имеет смысл разбить на три горизонтальных фрагмента:

C1 = C ►tabNo E1
C2 = C ►tabNo E2
C3 = C ►tabNo E3

где символ ► обозначает естественное полусоединение отношения C и фрагмента Ei (включает кортежи отношения C, которые могут быть соединены с соответствующим кортежем фрагмента Ei по значению внешнего ключа).

Репликация данных

Репликация – это поддержание двух и более идентичных копий (реплик) данных на разных узлах РБД.

Реплика может включать всю базу данных (полная репликация), одно или несколько взаимосвязанных отношений или фрагмент отношения.

Достоинства репликации:

– повышение доступности и надежности данных;

– повышение локализации ссылок на реплицируемые данные.

Недостатки репликации:

– сложность поддержания идентичности реплик;

– увеличение объема памяти для хранения данных.

Поддержание идентичности реплик называется **распространение изменений** и реализуется **службой тиражирования**.

Служба тиражирования

Служба тиражирования должна выполнять следующие функции:

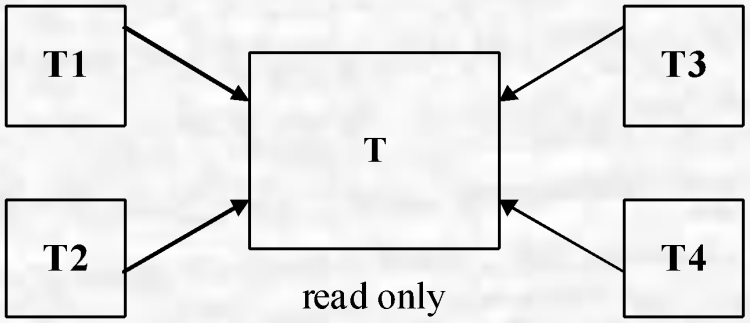
- ✔ Обеспечение масштабируемости, т.е. эффективной обработки больших и малых объемов данных.
- ✔ Преобразование типов и моделей данных (для гетерогенных РБД).
- ✔ Репликация объектов БД, например, индексов, триггеров и т.п.
- ✔ Инициализация вновь создаваемой реплики.
- ✔ Обеспечение возможности "подписаться" на существующие реплики, чтобы получать их в определенной периодичностью.

Для выполнения этих функций в языке, поддерживаемом СУБД, предусматривается наличие средств определения схемы репликации, механизма подписки и механизма инициализации реплик (создания и заполнения данными).

Репликация с основной копией

Существуют следующие варианты:

1. **Классический подход** заключается в наличии одной основной копии, в которую можно вносить изменения; остальные копии создаются с определением read only.
2. **Асимметричная репликация**: основная копия фрагментирована и распределена по разным узлам РБД, и другие узлы могут являться подписчиками отдельных фрагментов (read only).
3. **Рабочий поток**. При использовании этого подхода право обновления не принадлежит постоянно одной копии, а переходит от одной копии в другой в соответствии с потоком операций. В каждый момент времени обновляться может только одна копия**.**
4. **Консолидация данных:**



Репликация без основной копии

Симметричная репликация (без основной копии). Все копии реплицируемого набора могут обновляться одновременно и независимо друг от друга, но все изменения одной копии должны попасть во все остальные копии.

Существует два основных механизма распространения изменений при симметричной репликации:

- **синхронный:** изменения во все копии вносятся в рамках одной транзакции;
- **асинхронный:** подразумевает отложенный характер внесения изменений в удаленные копии.

Достоинство синхронного распространения изменений – полная согласованность копий и отсутствие конфликтов обновления.

Недостатки:

– трудоемкость и большая длительность модификации данных,

– низкая надежность работы системы.

Репликация без основной копии

Конфликтные ситуации:

- Добавление двух записей с одинаковыми первичными или уникальными ключами. Для предотвращения таких ситуаций обычно каждому узлу РБД выделяется свой диапазон значений ключевых (уникальных) полей.
- Конфликты удаления: одна транзакция пытается удалить запись, которая в другой копии уже удалена другой транзакцией. Если такая ситуация считается конфликтом, то она разрешаются вручную.
- Конфликты обновления: две транзакции в разных копиях обновили одну и ту же запись, возможно, по-разному, и пытаются распространить свои изменения. Для идентификации конфликтов обновления необходимо передавать с транзакцией дополнительную информацию: старое и новое содержимое записи. Если старая запись не может быть обнаружена, налицо конфликт обновления.

Репликация без основной копии

Методы разрешения конфликтов обновления:

1. Разрешение **по приоритету узлов**: для каждого узла назначается приоритет, и к записи применяется обновление, поступившее с узла с максимальным приоритетом.
2. Разрешение **по временной отметке**: все транзакции имеют временную отметку, и к записи применяется обновление с минимальной или максимальной отметкой. Использовать ли для этого минимальную или максимальную отметку – зависит от предметной области и, обычно, может регулироваться.
3. **Аддитивный метод** (add – добавить): может применяться в тех случаях, когда изменения основаны на предыдущем значении поля, например, salary = salary + X. При этом к значению поля последовательно применяются все обновления.
4. Использование **пользовательских процедур**.
5. Разрешение конфликтов **вручную**. Сведения о конфликте записываются в журнал ошибок для последующего анализа и устранения администратором.

Репликация без основной копии

Способы реализации распространения изменений:

1. Использование триггеров.
   Внутрь триггера помещаются команды, проводящие на других копиях обновления, аналогичные тем, которые вызвали выполнение триггера.
   Этот подход достаточно гибкий, но он обладает рядом недостатков:
   - триггеры создают дополнительную нагрузку на систему;
   - триггеры не могут выполняться по графику (время срабатывания триггера не определено);
   - с помощью триггеров сложнее организовать групповое обновление связанных таблиц (из-за проблемы мутирующих таблиц).
2. Поддержка журналов изменений для реплицируемых данных. Рассылка этих изменений входит в задачу сервера СУБД или сервера тиражирования (входящего в состав СУБД). Основные принципы, которых необходимо придерживаться при этом:
   - Для сохранения согласованности данных должен соблюдаться порядок внесения изменений.
   - Информация об изменениях должна сохраняться в журнале до тех пор, пока не будут обновлены все копии этих данных.

Распределенные запросы

Распределенным называется запрос, который обращается к двум и более узлам РБД, но не обновляет на них данные.

Запрашивающий узел должен определить, что в запросе идет обращение к данным на другом узле, выделить подзапрос к удаленному узлу и перенаправить его этому узлу.

Самой сложной проблемой выполнения распределенных запросов является **оптимизация**, т.е. поиск оптимального плана выполнения запроса. Информация, которая требуется для оптимизации запроса, распределена по узлам. Если выбрать центральный узел, который соберет эту информацию, построит оптимальный план и отправит его на выполнение, то теряется свойство локальной автономности.

Поэтому обычно распределенный запрос выполняется так: запрашивающий узел собирает все данные, полученные в результате выполнения подзапросов, у себя, и выполняет их соединение (или объединение), что может занять очень много времени.

Распределенные запросы. Пример

База данных "Агентство недвижимости", 2 филиала – в Лондоне и Глазго. Отношения:

Property (pNo, City, ...), 10 000 записей, хранится в Лондоне.

Renter (rNo, Max_price, ...), 100 000 записей, хранится в Глазго.

Viewing (pNo, rNo), 1 000 000 записей, хранится в Лондоне.

Время передачи = C_0 + (количество байт) / (скорость передачи), C_0 =1с

Запрос: список объектов в Абердине, которые осмотрены потенциальными покупателями, согласными заплатить не менее 200000.

```
select p.pNo
from property p, renter r, viewing v
where    p.pNo=v.pNo and r.rNo=v.rNo and
      p.City='Aberdine' and r.Max_price>=200000;
```

Распределенные запросы. Пример

Условия:

- ✔ скорость передачи 10000 б/с;
- ✔ задержка передачи – 1 с,
- ✔ все кортежи по 100 байт,
- ✔ существует 10 покупателей, согласных заплатить не менее 200000,
- ✔ в Абердине было проведено 100000 осмотров.

Ротни (Rothnie) проанализировал 6 стратегий выполнения этого запроса:

1. Переслать Renter в Лондон и выполнить обработку запроса там: 1+(100000*100)/10000 = **16,7 мин.**
2. Переслать Viewing и Property в Глазго и выполнить обработку запроса там: 2+((1000000+10000)*100)/10000 = **28 ч**
3. Соединить Renter и Property в Лондоне и для каждого кортежа проверить покупателя: 100000*(2+100/10000) +1*100000 = **2,3 дня**
4. Выбрать в Глазго нужных покупателей и проверить для каждого город: 10*(1+100/10000) +1*10 = **20 с**
5. Соединить Renter и Property в Лондоне, выполнить проекцию полей pNo и rNo и переслать её в Глазго: 1+(100000*100)/10000 = **16,7 мин.**
6. Выбрать клиентов по Max_price и переслать в Лондон: 1+(10*100)/10000 = **1 с**

Распределенные ограничения целостности

Распределенные ограничения целостности возникают тогда, когда для проверки соблюдения какого-либо ограничения целостности системе необходимо обратиться к другому узлу.

Примеры:

1) База данных разделена на фрагменты таким образом, что родительская таблица находится на одном узле, а дочерняя, связанная с ней по внешнему ключу, – на другом. При добавлении записи в дочернюю таблицу система обратится к узлу, на котором расположена родительская таблица, для проверки наличия соответствующего значения ключа.
2) Разбиение одной таблицы на фрагменты и размещение этих фрагментов по разным узлам сети. Здесь будет необходима проверка соблюдения ограничений первичного ключа и уникальных ключей.

Распределенные транзакции

Распределенные транзакции обращаются к двум и более узлам и обновляют на них данные.

Основная проблема распределенных транзакций – соблюдение логической целостности данных. Транзакция на всех узлах должна завершиться одинаково: или фиксацией, или откатом.

Выполнение распределенных транзакций осуществляется с помощью специального алгоритма, который называется **двухфазная фиксация**.

Координатор транзакции – узел, который контролирует выполнение этого протокола (обычно, тот узел, который инициирует данную транзакцию).

Остальные узлы, на которых выполняется транзакция, называются ***участниками транзакции***.

Протокол двухфазной фиксации

а) диаграмма состояний координатора

б) диаграмма состояний участника

Действия координатора транзакции

Координатор выполняет протокол 2ФФ по следующему алгоритму:

I. Фаза 1 (голосование).

Занести запись *begin_commit* в системный журнал и обеспечить ее перенос из буфера в ОП на ВЗУ. Отправить всем участникам команду PREPARE.

Ожидать ответов всех участников в пределах установленного тайм-аута.

II. Фаза 2 (принятие решения).

При поступлении сообщения ABORT: занести в системный журнал запись *abort* и обеспечить ее перенос из буфера в ОП на ВЗУ; отправить всем участникам сообщение GLOBAL_ABORT и ждать ответов участников (тайм-аут).

Если участник не отвечает в течение установленного тайм-аута, координатор считает, что данный участник откатит свою часть транзакции и запускает протокол ликвидации.

Если все участники прислали COMMIT, поместить в системный журнал запись *commit* и обеспечить ее перенос из буфера в ОП на ВЗУ. Отправить всем участникам сообщение GLOBAL_COMMIT и ждать ответов всех участников.

После поступления подтверждений о фиксации от всех участников: поместить в системный журнал запись *end_transaction* и обеспечить ее перенос из буфера в ОП на ВЗУ.

Если некоторые узлы не прислали подтверждения фиксации, координатор заново направляет им сообщения о принятом решении и поступает по этой схеме до получения всех требуемых подтверждений.

Действия участника транзакции

Участник выполняет протокол 2ФФ по следующему алгоритму:

1. При получении команды PREPARE, если он готов зафиксировать свою часть транзакции, он помещает запись *ready_commit* в файл журнала транзакций и отправляет координатору сообщение READY_COMMIT. Если он не может зафиксировать свою часть транзакции, он помещает запись *abort* в файл журнала транзакций, отправляет координатору сообщение ABORT и откатывает свою часть транзакции (не дожидаясь общего сигнала GLOBAL_ABORT).
2. Если участник отправил координатору сообщение READY_COMMIT, то он ожидает ответа координатора в пределах установленного тайм-аута.
3. При получении GLOBAL_ABORT участник помещает запись *abort* в файл журнала транзакций, откатывает свою часть транзакции и отправляет координатору подтверждение отката.
4. При получении GLOBAL_COMMIT участник помещает запись *commit* в файл журнала транзакций, фиксирует свою часть транзакции и отправляет координатору подтверждение фиксации.
5. Если в течение установленного тайм-аута участник не получает сообщения от координатора, он откатывает свою часть транзакции.

Протоколы ликвидации

Протокол ликвидации для координатора:

1. Тайм-аут в состоянии WAITING: координатор не может зафиксировать транзакцию, потому что не получены все подтверждения от участников о фиксации. Ликвидация заключается в откате транзакции.
2. Тайм-аут в состоянии DECIDED: координатор повторно рассылает сведения и принятом глобальном решении и ждет ответов от участников.

Простейший протокол ликвидации для участника заключается в блокировании процесса до тех пор, пока сеанс связи с координатором не будет восстановлен. Но в целях повышения производительности (и автономности) узлов могут быть предприняты и другие действия:

1. Тайм-аут в состоянии INITIAL: участник не может сообщить о своем решении координатору и не может зафиксировать транзакцию. Но может откатить свою часть транзакции. Если он позднее получит команду PREPARE, он может проигнорировать ее или отправить координатору сообщение ABORT.
2. Тайм-аут в состоянии PREPARED: участник уже известил координатор о решении COMMIT, то он не может его изменить. Участник оказывается заблокированным.

Протоколы восстановления

Действия, которые выполняются на отказавшем узле после его перезагрузки, называются *протоколом восстановления*.

Они зависят от того, в каком состоянии находился узел, когда произошел сбой, и какую роль выполнял этот узел в момент отказа: координатора или участника.

При отказе координатора:

- ✔ В состоянии INITIAL: процедура 2ФФ еще не запускалась, поэтому после перезагрузки следует ее запустить.
- ✔ В состоянии WAITING: координатор уже направил команду PREPARE, но еще не получил всех ответов и не получил ни одного сообщения ABORT. В этом случае он перезапускает процедуру 2ФФ.
- ✔ В состоянии DECIDED: координатор уже направил участникам глобальное решение. Если после перезапуска он получит все подтверждения, то транзакция считается успешно зафиксированной. В противном случае он должен прибегнуть к протоколу ликвидации.

Протоколы восстановления

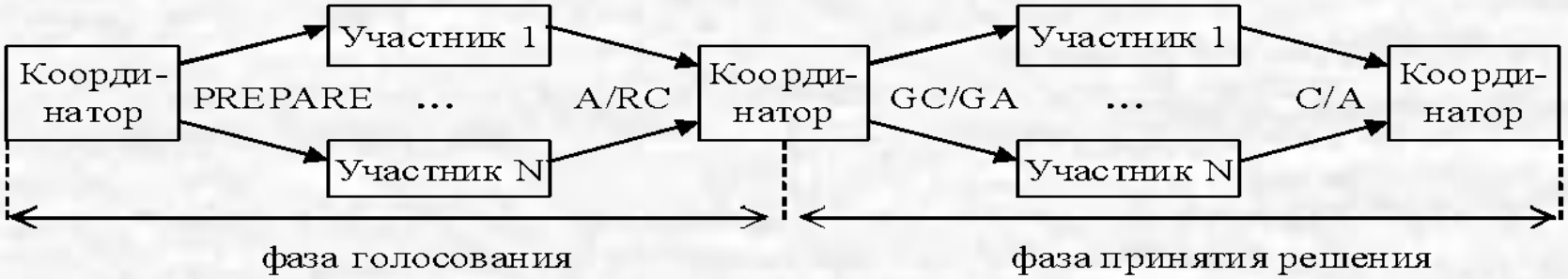
При отказе участника цель протокола восстановления – гарантировать, что после восстановления узел выполнит в отношении транзакции то же действие, которое выполнили другие участники, и сделает это независимо от координатора, т.е. по возможности без дополнительных подтверждений.

Рассмотрим три возможных момента возникновения отказа:

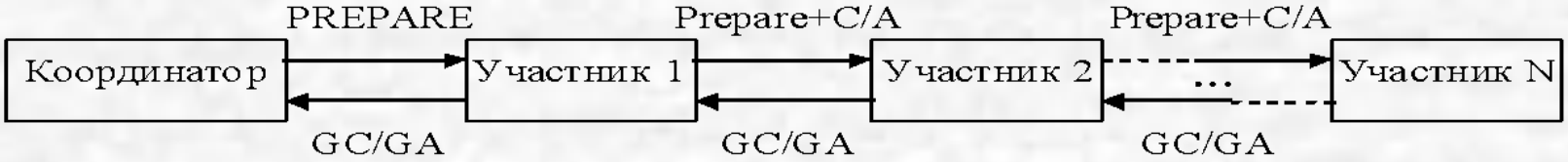
- ✔ В состоянии INITIAL: участник еще не успел сообщит о своем решении координатору, поэтому он может выполнить откат, т.к. координатор не мог принять решение о глобальной фиксации транзакции без голоса этого участника.
- ✔ В состоянии PREPARED: участник уже направил сведения о своем решении координатору, поэтому он должен запустить свой протокол ликвидации.
- ✔ В состоянии ABORTED/COMMITED: участник уже завершил обработку своей части транзакции, поэтому никаких дополнительных действий не требуется.

Реализация протокола 2ФФ

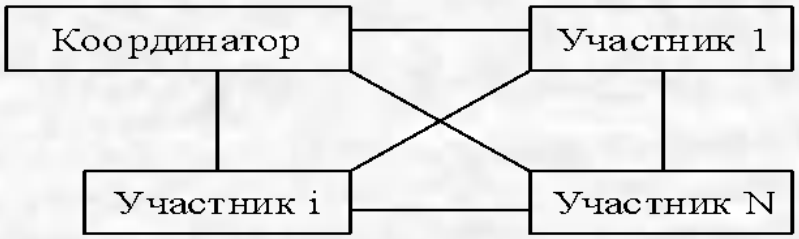
а) традиционная реализация протокола 2ФФ



б) линейная реализация протокола 2ФФ



в) распределенная реализация протокола 2ФФ



Поддержка распределенности в Oracle

1. Прозрачность распределенности.
2. Каждая часть данных, хранимых на одном компьютере в сети, оформлена как самостоятельная база данных.
3. Одна логическая таблица может быть распределена по разным узлам в сети.
4. Каждый сервер БД в системе распределенной базы данных (РБД) управляет доступом к своей локальной БД; за управление системой в целом не отвечает ни один сервер.
5. Поддержка целостности и согласованности данных осуществляется на уровне взаимодействия между серверами, что является расширением модели клиент-сервер (Distributed Oracle).
6. Связь осуществляется по сети с помощью программного средства Oracle – дополнительной утилиты Net8.
7. Распределенная БД может быть неоднородной, при этом один или несколько узлов должны быть Oracle-серверами, а связь с серверами других типов осуществляется через открытый шлюз (дополнительный программный пакет Open Gateways).

Связь в распределенной БД Oracle

Обращение к сервисам базы данных (серверу БД, очереди печати, серверу электронной почты и т.д.) происходит по уникальному имени (global name). Для БД оно состоит из основного имени $ORACLE_SID, назначаемого ей при создании (длиной не более 8-и символов) и сетевого домена БД, например:

sales.parts@east.compworld

– обращение к таблице PARTS базы данных SALES, расположенной на сервере east.compworld.

Для получения доступа к удаленной БД нужно установить связь с этой БД с помощью специальной команды языка SQL – LINK. При формировании связи могут учитываться учетные сведения пользователя для обеспечения безопасности данных, но это требует дополнительных усилий для распространения учетных сведений пользователя в сети: во-первых, нужно создать учетный раздел пользователя на удаленном сервере; во-вторых, пакеты регистрации в сети желательно шифровать, т.к. сеть не защищена от доступа посторонних лиц.

Связи в распределенной БД Oracle

Примеры.

Локальная база данных – HQ.ACME.COM.

Удаленная база данных – SALES.ACME.COM.

Создание общей связи баз данных к удаленной базе данных SALES:

CREATE PUBLIC DATABASE LINK sales.acme.com USING 'dbstring';

Создание личной связи баз данных для создателя этой связи:

CREATE DATABASE LINK sales CONNECT TO scott IDENTIFIED BY tiger;

Фраза CONNECT TO специфицирована явно. При установлении сессии в удаленной базе данных через эту связь баз данных будет использоваться идентификация SCOTT/TIGER.

Фраза USING опущена. Поэтому, когда используется эта личная связь баз данных, должна существовать одноименная общая или сетевая связь баз данных, содержащая строку базы данных для установления соединения с удаленной базой данных.

Работа в распределенной БД Oracle

Oracle различает следующие виды обработки данных в РБД:

удаленный запрос – это оператор SELECT, считывающий информацию из одной или нескольких таблиц на одном из удаленных узлов сети;

распределенный запрос – это оператор SELECT, считывающий информацию из одной или нескольких удаленных таблиц, которые расположены на разных узлах;

удаленное обновление – это модификация, затрагивающая таблицы из одного удаленного узла;

распределенное обновление – это модификация данных двух или более узлов сети;

вызовы удаленных процедур – запуск процедуры, находящейся на удаленном сервере;

удаленная транзакция – это транзакция, содержащая хотя бы один удаленный запрос и относящаяся к одному удаленному узлу;

распределенная транзакция – это транзакция, содержащая одно или несколько распределенных обновлений или удаленные транзакции для разных серверов. За выполнение распределенных транзакций отвечает механизм двухфазной фиксации.

Моментальные снимки в Oracle

Oracle поддерживает два типа тиражирования:

- ✔ базовое – копия обеспечивает доступ "только для чтения".
- ✔ усовершенствованное – приложения могут считывать и обновлять тиражируемые копии таблиц по всей системе (поддерживается специальными средствами СУБД – **Replication Option**).

Базовое тиражирование осуществляется (после установления связи с удаленной БД) с помощью создания моментальных снимков (snapshot), например:

```
CREATE SNAPSHOT sales.parts AS
        SELECT * FROM sales.parts@central.compworld;
```

Моментальные снимки бывают:

- ✔ простые – создаются по однотабличному запросу SELECT, содержащему простые условия отбора.
- ✔ сложные – создаются по запросам, содержащим сложные условия отбора, фразы group by, having, обращающимся к двум и более таблицам и проч.

Моментальные снимки в Oracle

Примеры:

Моментальный снимок, основой которого является запрос

```
select * from employee@hr_link;
```

является простым.

Моментальный снимок, основанный на запросе

```
select dept, max(salary)
    from employee@hr_link
    group by dept;
```

сложный, так как в нем используются функции группирования.

С помощью моментального снимка в локальной базе данных будет создано несколько объектов, поэтому пользователь, создающий моментальный снимок, должен иметь привилегии CREATE TABLE, CREATE VIEW и CREATE INDEX.

Моментальные снимки в Oracle

Синтаксис создания моментального снимка:

create snapshot [*имя_схемы*.]*имя_снимка*
[{ pctfree *целое* | pctused *целое* | initrans *целое* |
maxtrans *целое* | tablespace *имя_табличной_области* |
storage *размер_памяти* }]
[cluster *имя_кластера* (*имя_столбца*[,…])]
[using index]
[{ pctfree *целое* | pctused *целое* | initrans *целое* |
maxtrans *целое* | tablespace *имя_табличной_области* |
storage *размер_памяти* }]
[refresh [{ fast | complete | force }]
[start with *дата_1*] [next *дата_2*]]
[for update]
as *запрос*;

Моментальные снимки в Oracle

Пример создания МС на локальном сервере:

```
create snapshot emp_dept_count
        pctfree 5
        tablespace snap
        storage (initial 100k next 100k pctincrease 0)
        refresh complete
        start with sysdate
        next sysdate+7
        as select deptno, count(*) dept_count
                from employee@hr_link
                group by deptno;
```

Моментальные снимки в Oracle

При создании моментального снимка в локальной базе данных создается:

- ✔ **таблица** для хранения записей, получаемых в результате выполнения запроса моментального снимка (с именем *SNAP$_имя_моментального_снимка)*;
- ✔ **представление** этой таблицы "только для чтения", называемое в соответствии с именем моментального снимка;
- ✔ **представление**, называемое MVIEW$_имя_моментального_снимка – для обращения к удаленной основной таблице (или таблицам). Это представление будет использоваться во время регенерации.

Для модификации снимка, например, с целью установки частоты автоматического изменения в 1 час можно воспользоваться командой ALTER SNAPSHOT:

```
alter snapshot emp_dept_count refresh complete
        start with sysdate next sysdate + 1/24;
```

Для удаления моментальных снимков применяется команда drop snapshot:

```
drop snapshot emp_dept_count;
```

Регенерация моментальных снимков Oracle

Возможны два варианта:

1. REFRESH FAST (**быстрая регенерация**).
2. REFRESH COMPLETE (**полная регенерация**).

Режим регенерации	Описание
COMPLETE	Таблицы моментального снимка полностью восстанавливаются с помощью его запроса и основных таблиц при каждой регенерации
FAST	Если применяется простой моментальный снимок, то для посылки только тех изменений, которые внесены в его таблицу, можно использовать журнал моментальных снимков
FORCE	Значение по умолчанию. Если это возможно, выполняется быстрая (FAST) регенерация, если нет – полная (COMPLETE) регенерация

Регенерация моментальных снимков Oracle

Для быстрой регенерации необходим **журнал моментальных снимков** (snapshot log) – это таблица, обеспечивающая регистрацию в моментальном снимке изменений, происшедших в основной таблице. Имя журнала (таблицы) – MLOG$_имя_таблицы.

Команда CREATE SNAPSHOT LOG. Пример:

```
create snapshot log on employee
        tablespace data
        storage (initial 10k next 10k pctincrease 0);
```

Изменения в журнал моментальных снимков попадают с помощью триггера AFTER типа FOR EACH ROW, который называется TLOG$_имя_таблицы.

В журнале моментальных снимков данные находятся очень непродолжительное время: записи вводятся в журнал моментальных снимков, используются во время регенерации, а затем удаляются из журнала автоматически.

Усовершенствованное тиражирование Oracle

Производится с помощью двух средств Oracle:

1. Многоабонентского тиражирования.
2. Узлов обновляемых моментальных снимков.

Распространение изменений:

1. на уровне строк: сервер записывает изменения, сделанные каждой DML-транзакцией, и рассылает эти изменения в удаленные узлы.
2. путем процедурного тиражирования: тиражируется вызов удаленной процедуры, выполняющей в удаленном узле те же изменения, что и в вызывающем.

Различают **асинхронное** и **синхронное** распространение изменений.

Внесение изменений в тиражируемые данные происходит в несколько этапов:

- ✔ локальный узел вносит изменения в свою копию данных (ОМС);
- ✔ локальный узел запускает отложенную транзакцию на основном узле;
- ✔ через некоторое время локальный узел выполняет быструю (или полную) регенерацию локальной копии данных, после чего приложение всегда может проверить, выполнена ли инициированная им транзакция. Если она не выполнена, то происходит рестарт транзакции и все повторяется.