### Лекция 3. Принципы функционирования систем с РБД

**1. Фундаментальный принцип РБД**

**2. Независимость от конфигурации**

**3. Независимость от системных средств**

**4. Независимость от данных**

**1. Фундаментальный принцип РБД**

Фундаментальным принципом распределенной базы данных формулируется следующим образом:

* *Для пользователя распределенная система должна выглядеть точно так же, как и* **нераспределенная** *система*

Работу пользователей в распределенной системе следует организовывать таким же образом, как если бы она не была распределенной. Все связанные с распределенными системами проблемы являются (или должны быть) внутренними и должны возникать только на внутреннем уровне или на уровне разработки, а не на внешнем уровне или на уровне пользователей.

В данном случае термин «пользователи» относиться к пользователям (потребителям или разработчикам приложений), которые выполняют операции управления данными. Все эти операции должны оставаться логически неизменными, в отличие от операций определения данных, которые, наоборот, могут быть расширены в распределенной системе. Например, пользователь на узле Х может указать, что хранимое отношение можно разделить на «фрагменты» для физического хранения на узлах Y и Z. Понятие «фундаментального принципа» было впервые предложено К. Дж. Дейтом и было названо «правилом нуль» (Rule Zero). Данный принцип приводит к набору следующих правил и целей:

|  |  |  |  |
| --- | --- | --- | --- |
| **Фундаментальный принцип (глобальная прозрачность)** | | | |
| **Независимость от конфигурации** | **Независимость от системных средств** | **Независимость от данных** | **Режимы работы** |
| 1. Локальная автономия | 3. Независимость от аппаратного обеспечения | 7. Независимость от расположения | 10. Непрерывное функционирование |
| 2. Независимость от центрального узла | 4. Независимость от операционной системы | 8. Независимость от фрагментации | 11. Обработка распределенных запросов |
|  | 5. Независимость от сети | 9. Независимость от репликации | 12. Управление распределенными транзакциями |
|  | 6. Независимость от СУБД |  |  |

Эти двенадцать целей не являются независимыми одна от другой, к тому же не все они равнозначны (различные пользователи могут придавать разное значение разным целям в разном окружении), ими также не исчерпывается список всех возможных целей. Однако они весьма полезны для понимания основ распределенной технологии и для общей характеристики функциональности некоторой распределенной системы.

Необходимо различать распределенные системы и системы, в которых поддерживаются некоторые способы удаленного доступа к данным . В системах с удаленным доступом к данным пользователь может одновременно работать с данными, расположенными на нескольких удаленных узлах, однако в таком случае будут видны соединения между ними. В истинной распределенной системе, наоборот, все соединения от пользователя скрыты.

**2. Независимость от конфигурации**

# Локальная автономия. В распределенной системе узлы следует делать автономными. Локальная автономия означает, что операции на данном узле управляются этим узлом, т.е. функционирование любого узла Х не зависит от успешного выполнения некоторых операций на каком-то другом узле У (иначе выход из строя узла У может привести к невозможности исполнения операций на узле X, даже если с узлом Х ничего не случилось) Из принципа локальной автономии также следует, что владение и управление данными осуществляется локально. Действительно, даже если доступ к данным осуществляется с других удаленных узлов, все они относятся к некоторой локальной базе данных. Такие вопросы, как безопасность, целостность и структура хранения локальных данных, остаются под контролем и локального узла.

В действительности цель локальной автономии достигается не полностью, поскольку есть множество ситуаций, в которых узел Х должен предоставить некоторую часть управления другому узлу У. Поэтому цель достижения локальной автономии требует более точной формулировки, а именно: узлы следует делать автономными в максимально возможной степени.

**Независимость от центрального узла.** Под локальной автономией подразумевается, что все **узлы должны рассматриваться как равные.** Следовательно, не должно существовать никакой зависимости и от центрального «основного» узла с некоторым централизованным обслуживанием, например централизованной обработкой запросов или централизованным управлением транзакциями. Таким образом, вторая цель является логическим следствием первой . Однако, если локальная автономия не достигается в полной мере, даже достижение независимости от центрального узла само по себе очень важно и может рассматриваться как отдельная цель.

Зависимость от центрального узла нежелательна по крайней мере по двум причинам. Во-первых, центральный узел может быть «узким» местом всей системы, а во-вторых, более важно то, что система в таком случае становиться очень уязвимой, т.е. при выходе из строя центрального узла может отказать вся система.

**3. Независимость от системных средств**

**Независимость от аппаратного обеспечения.** Здесь трудно добавить что-либо существенное, поскольку основная идея фактически полностью изложена в названии этого раздела. Используемые в настоящее время компьютеры характеризуются весьма большим разнообразием, среди них можно встретить компьютеры фирм IBC, DEC, HP, а также персональные компьютеры и рабочие станции других типов. В связи с этим существует реальная необходимость интеграции данных на всех этих системах и создания для пользователя «представления единой системы». Таким образом, весьма важной является возможность запуска копий одной и той же СУБД на разном аппаратном обеспечении с тем, чтобы разные компьютеры могли работать в распределенной система как равные партнеры.

**Независимость от операционной системы.** Эта цель является следствием предыдущей, и ее описание будет также немногословным. Очевидно, важно запустить одну и ту же СУБД не только на разном аппаратном обеспечении, но и на разных операционных системах, причем даже в тех случаях, когда разные операционные системы используются на однотипном аппаратном обеспечении. Например, для того чтобы версии СУБД для операционной системы MVS, а также для систем UNIX и PC/DOS могли совместно работать в одной и той же распределенной системе.

**Независимость от сети.** Здесь также можно ограничиться кратким замечанием о том, что если система в состоянии поддерживать несколько узлов с разным аппаратным обеспечением и разными операционными системами, то было бы желательно, чтобы в ней поддерживались также разные типы сетей.

**Независимость от СУБД.** Эта цель подразумевает использование несколько менее точной формулировки предположения о строгой однородности. В новой форме это предположение означает, что **все экземпляры СУБД на различных узлах поддерживают один и тот же интерфейс**, хотя они не обязательно должны быть копиями одного и того же программного обеспечения. Например, если бы системы INGRES и ORACLE поддерживали официальный стандарт SQL, то *было бы возможно* организовать их совместную работу в контексте распределенной системы. Иначе говоря, распределенная система, по крайней мере в некоторой степени, может быть неоднородной.

Поддержка этой неоднородности весьма желательна. Дело в том, что в реальном мире работа обычно организована не только на компьютерах разных типов и в разных операционных системах, но и с участием различных СУБД. Было бы прекрасно, если бы такие СУБД могли работать совместно в одной распределенной системе. Иначе говоря, в идеальной распределенной системе предполагается поддержка **независимости от СУБД**.

Этой довольно обширной и очень важной с практической точки зрения теме посвящен отдельный раздел.

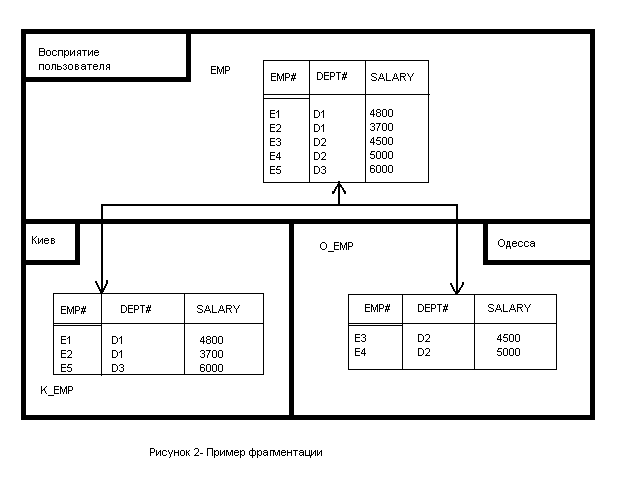
**4 Независимость от данных**

## 4.1 Независимость от расположения . Основная идея независимости от расположения (которая также называется прозрачностью расположения) достаточно проста: пользователям не следует знать, в каком физическом месте хранятся данные, наоборот, с логической точки зрения пользователям следовало бы обеспечить такой режим, при котором создается впечатление, что все данные хранятся на их собственном локальном узле. Обеспечить независимость от расположения весьма желательно, поскольку при этом существенно упрощаются пользовательские программы и терминальная деятельность. В частности, это позволяет осуществлять миграцию данных от узла к узлу без объявления недействительными любых пользовательских программ и видов терминальной деятельности. Процесс миграции весьма полезен, поскольку позволяет перемещаться данным по всей сети в ответ на изменение требований к производительности.

**4.2 Независимость от фрагментации.**

**Фрагментация данных** означает что данные разделяются на части таким образом чтобы части (физические фрагменты данных - таблицы ) хранились в том месте где они чаще используются. При такой организации многие операции будут чисто локальными, а объём пересылаемых в сети данных снизится. Однако с логической точки зрения пользователю обеспечивается такой режим работи при котором данные кажутся нефрагментированы вовсе.

Фрагментация желательна для повышения производительности системы, поскольку данные лучше хранить в том месте, где они наиболее часто используются. При такой организации многие операции будут чисто локальными, а объем пересылаемых в сети данных снизится. Рассмотрим отношение сотрудников ЕМР, показанное на рис 2. В системе, в которой поддерживается фрагментация, определены два фрагмента, показанных на рис.2



Теперь оптимизатору известно, что доступ требуется осуществить только к узлу в Киеве.

Как следует из сказанного выше, проблема поддержки операций для фрагментированных отношений имеет некоторое сходство с проблемой поддержки операций для объединенных и соединенных представлений (фактически, это разные проявления одной и той же проблемы при рассмотрении общей архитектуры системы с разных точек зрения). В частности проблема обновления фрагментированных отношений - это проблема обновления в объединениях и соединениях представлений. Отсюда также следует, что при обновлении заданный кортеж может мигрировать из одного фрагмента в другой при условии, что он больше не удовлетворяет предикату отношения для фрагмента, к которому относился до обновления.

Существует два основных типа фрагментации – горизонтальная и вертикальная, которые связаны с реляционными операциями выборки и проекции соответственно. (На рис. 21.2 показана горизонтальная фрагментация.) Иначе говоря, фрагментом может быть *любое произвольное подчиненное отношение*, которое можно вывести из исходного отношения с помощью операций выборки и проекции. При этом следует учесть приведенные ниже допущения.

* Предполагается без утраты общности, что все фрагменты данного отношения независимы, т.е. ни одни из фрагментов не может быть выведен из других фрагментов либо иметь выборку или проекцию, которая может быть выведена из других фрагментов. Проекции не должны допускать потерю информации.

Реконструкцию исходного отношения на основе его фрагментов можно осуществить с помощью операций соединения (для вертикальных фрагментов) и объединения (для горизонтальных фрагментов). Обратите внимание, что в случае объединения не потребуется выполнять операцию исключения дубликатов (объясните, почему).

*Замечание*. В отношении вертикальной фрагментации необходимо привести дополнительные разъяснения. Как уже упоминалось выше, при такой фрагментации не должна допускаться потеря информации, потому что фрагментация отношения ЕМР на проекции (EMP#, DEPT#) и (SALARY) не будет достоверной. Однако в некоторых системах хранимые отношения имеют скрытый атрибут TID (“tuple ID” – идентификатор кортежа), т.е. физический или логический адрес этого кортежа. Атрибут TID является потенциальным ключом для данного кортежа, поэтому если бы отношение ЕМР содержало такой атрибут, то *оно могло бы* быть корректно фрагментировано на проекции (TID, EMP#, DEPT#) и (TID, SALARY), поскольку такая фрагментация происходит без потери информации.

Обратите внимание, что именно легкость выполнения фрагментации и реконструкции – одна из многих причин, по которым для распределенных систем используется реляционная модель. Наконец, переходя к главному, нужно сказать, что в системе, поддерживающей фрагментацию данных, следует также предусмотреть поддержку **независимости от фрагментации** (или **прозрачность** фрагментации). Иными словами, с логической точки зрения пользователям следует обеспечить такой режим работы, при котором данные кажутся вовсе не фрагментированными. Независимость от фрагментации (как и независимость от расположения) весьма желательна, поскольку она позволяет в любой момент осуществить дефрагментацию данных (и фрагменты могут быть перераспределены в любой момент времени) в ответ на изменение требований к производительности, причем без необходимости отключения любых пользовательских программ и терминальной деятельности.

Независимость от фрагментации предполагает, что данные будут представлены для пользователей в виде логически комбинированных фрагментов на основе соответствующих объединений и соединений. При этом системный оптимизатор отвечает за определение фрагментов, к которым необходимо обеспечить физический доступ для выполнения любого заданного пользователем запроса. В качестве примера допустим, что для фрагментации, показанной на рис. 21.2, пользователь задает следующий запрос:

EMP WHERE SALARY > 40K AND DEPT# = ‘D1’

В таком случае оптимизатор системы будет знать из определений фрагментов (которые хранятся в системном каталоге), что так как все результаты могут быть получены на узле в Нью-Йорке, то нет никакой необходимости осуществлять доступ к узлу в Лондоне.

Рассмотрим этот же пример более тщательно. Отношение ЕМР воспринимается пользователем как **представление** базовых фрагментов N\_EMP и L\_EMP: EMP = N\_EMP UNION L\_EMP

Таким образом, оптимизатор преобразует исходный запрос пользователя в следующий:

(N\_EMP UNION L\_EMP) WHERE SALARY > 40K AND DEPT# = ‘D1’

А это выражение, в свою очередь, можно преобразовать к виду

(N\_EMP WHERE SALARY > 40K AND DEPT# = ‘D1’) UNION

(L\_EMP WHERE SALARY > 40K AND DEPT# = ‘D1’)

Из хранящихся в каталоге определения фрагмента L\_EMP оптимизатору известно о том, что второе из этих двух выражений даст результат пустое отношение (условие целостности DEPT# = ‘D1’ AND DEPT# = ‘D2’ не может быть оценено как истинное). Таким образом, все выражения в целом можно свести к следующему упрощенному виду:

N\_EMP WHERE SALARY > 40K AND DEPT# = ‘D1’

Теперь оптимизатору известно, что доступ требуется осуществить только к узлу в Нью-Йорке.

Как следует из сказанного выше, проблема поддержки операций для фрагментированных отношений имеет некоторое сходство с проблемой поддержки операций для объединенных и соединенных представлений (фактически, это разные проявления одной и той же проблемы при рассмотрении общей архитектуры системы с разных точек зрения). В частности, проблема обновления фрагментированных отношений – это проблема обновления в объединениях и соединениях представлений. Отсюда также следует, что при обновлении заданный кортеж может мигрировать из одного фрагмента в другой при условии, что он больше не удовлетворяет предикату отношения для фрагмента, к которому относился до обновления.

## 4.3Независимость от репликации

**Репликация данных.** Репликация данных означает что некоторый фрагмент данних (таблица) может быть представлен в базе данных различными копиями (репликами) хранимыми на нескольких различных узлах.

Репликации полезны по двум причинам:

Увеличение производительности системы, так как приложения могут работать с локальными данными без обмена информацией с удаленными узлами

Увеличение доступности и надежности, так как объект остается доступным пока имеется хотя бы одна его реплика. Главный недостаток репликации заключается в том что при обновлении реплицируемого объекта все копии этого объекта должны также обновляться. С логической точки зрения пользователи работают таким образом как будто данные не реплицированы вовсе.

Существует несколько механизмов выполнения репликации, это - множественная публикация, централизованная публикация и централизованная подписка.

Ниже приведен пример репликации, схематически показанный на рис. 3.

REPLICATE К\_ЕМР ОК\_ЕМР АТ SIТЕ 'Одесса'; REPLICATE 0\_ЕМР КО\_ЕМР АТ SIТЕ 'Киев'; Внутренние имена реплик ОК\_ЕМР и КО\_ЕМР.

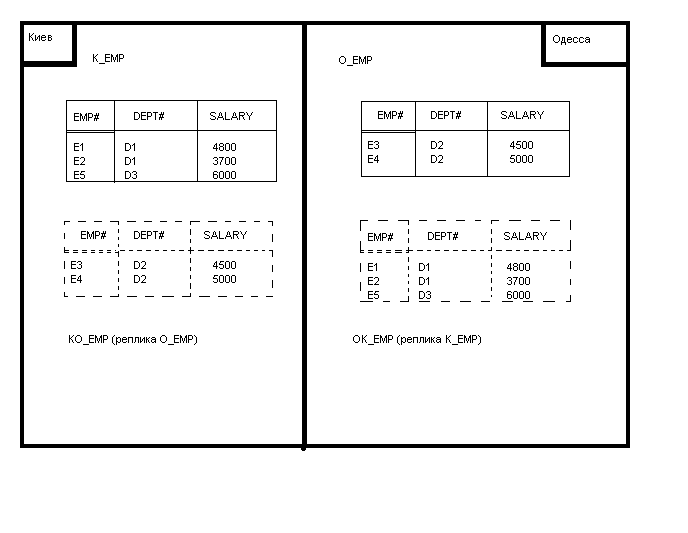


Рисунок 3- Пример репликации

Репликация полезна по крайней мере по двум причинам. Во-первых, благодаря ей достигается большая производительность (приложения могут работать с локальными копиями, не обмениваясь данными с другими узлами). Во-вторых, репликация также позволяет обеспечить большую доступность (реплицированный объект остается доступным для обработки до тех пор, пока остается хотя бы одна его реплика).

Главный недостаток репликации заключается в том, что при обновлении заданного реплицируемого объекта, все копии этого объекта также должны обновляться. Этот недостаток называется проблемой **распространения обновления**.

Следует отметить, что репликация в распределенной системе представляет собой особый вид воплощения идеи управляемой избыточности. В идеальном случае репликация, как и фрагментация, должна быть «прозрачной для пользователя». Иначе говоря, в системе, в которой поддерживается репликация данных, должна также поддерживаться **независимость от репликации** (или прозрачность репликации), т.е. пользователи, по крайней мере, с логической точки зрения, должны работать в таком режиме, как будто данные не реплицированы вовсе. Независимость от репликации (так же как независимость от расположения и независимость от фрагментации) весьма желательна, поскольку позволяет существенно упростить пользовательские программы и терминальную деятельность. В частности, при этом разрешается в любой момент создавать и удалять реплики в ответ на изменения требований без отключения этих пользовательских программ и терминальной деятельности.

Независимость от репликации подразумевает, что системный оптимизатор отвечает за определение физического доступа именно к тем репликам, которые необходимы для удовлетворения заданного пользовательского запроса.

**Пример реализации репликации.**

База данных "Российские ВУЗы" имеет один главный сервер, находящийся в Москве и нескольких региональных серверов. Московским сервером является MS SQL Server 6.5 под управлением Windows NT 4.0. Региональными серверами являются либо, так же, MS SQL Server 6.5 под управлением Windows NT 4.0, либо Oracle 7(8) под управлением Windows NT 4.0 или Linux.

В РБД применяется сценарий централизованной подписки. Администратор назначает сервер репликации и сервер подписки, определяет время, через которое будет происходить репликация и набор таблиц, участвующих в ней. Так же определяется несколько дополнительных параметров.

В нашем случае репликация происходит сразу после завершения транзакции обновления таблицы. Администратор с клиентского места вносит изменения в таблицу, используя для этого структурированный язык запросов SQL. После выполнения команды COMMIT WORK, означающей, что транзакция успешно завершена, сервер публикации/репликации устанавливает соединение с сервером подписки и проверяет данные из таблиц, участвующих в репликации, на идентичность. И в случае обнаружения новых данных сервер публикации реплицирует новые записи в соответствующие таблицы сервера подписки. В случае возникновения ошибки в процессе репликации (например, в случае разрыва сети) происходит откат выполнения репликации, и после восстановления работоспособности сети, система автоматически переходит в нормальное состояние и процесс репликации повторяется.

##### Тиражирование - частный случай репликации ( на примере INFORMIX-OnLine DS )

Тиражирование - это поддержание на другой вычислительной установке копии объектов базы данных. При этом реализуется прозрачное тиражирование данных с основного сервера баз данных на вторичный (или поддерживающий) сервер, к которому разрешен доступ только на чтение и который может находиться в другом географическом пункте. В этой терминологии сервер, не участвующий в тиражировании, называется стандартным.

Главная цель тиражирования в - это обеспечение высокой готовности (High Availability Data Replication - HDR). В случае отказа основного сервера вторичному серверу автоматически или вручную придается статус стандартного, с доступом на чтение и запись. Прозрачное перенаправление клиентов при отказе основного сервера не поддерживается, но оно может быть реализовано в рамках приложений. После восстановления основного сервера, в зависимости от значения параметра конфигурации, выбирается один из двух возможных сценариев:

* Восстановленному серверу вновь придается статус основного. Вторичный сервер, перед возвращением в режим доступа только на чтение, останавливается, чтобы обеспечить отсоединение клиентов, которые осуществляли доступ на запись.
* Восстановленный сервер становится вторичным, а бывшему вторичному, который уже функционирует в режиме чтения-записи, придается статус основного; клиенты, которые подключены к нему, продолжают работу. Этот сценарий обеспечивает непрерывную доступность баз данных.

Тиражирование осуществляется путем передачи информации из журнала транзакций (логического журнала) в буфер тиражирования основного сервера, откуда она пересылается в буфер тиражирования вторичного сервера. Такая пересылка может происходить либо в синхронном, либо в асинхронном режиме. Синхронный режим гарантирует полную согласованность баз данных - ни одна транзакция, зафиксированная на основном сервере, не останется незафиксированной на вторичном, даже в случае сбоя основного сервера. Асинхронный режим не обеспечивает абсолютной согласованности, но улучшает рабочие характеристики системы.

Зеркалирование, которое также является прозрачным средством поддержания высокой готовности, обеспечивает только копирование дисковых областей в пределах одной установки сервера INFORMIX-OnLine DS и защищает только от дисковых сбоев. Механизм тиражирования обеспечивает поддержание полной удаленной копии баз данных и защищает от всех видов отказов, включая полный крах одной из установок.

Помимо обеспечения отказоустойчивой работы, тиражирование дает следующие преимущества:

более оперативный доступ к данным для локальных клиентов вторичного сервера;

возможность вынести приложения пользователя преимущественно на вторичный сервер, где они выполняются с максимальным использованием данных, не подавляя приложений , выполняющихся на основном сервере.