Распределенные базы данных

Методы поддержки распределенных баз данных

Методы поддержки распределенных данных

1. **Фрагментация** – разбиение БД или таблицы базы данных 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. Может комбинироваться с фрагментацией.
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 базы данных

Фрагментация базы данных – основной способ организации РБД.
Назначение: хранение данных на том узле, где они чаще используются.
Основная проблема проектирования РБД – распределение данных по узлам с соблюдением следующих критериев:

- Локализация ссылок;
- Повышение доступности и надежности;
- Приемлемый уровень производительности;
- Компромисс между емкостью и стоимостью внешней памяти;
- Минимизация расходов на передачу данных.

Основные проблемы, которые возникают при использовании фрагментации БД:

– прозрачность написания запросов к данным;
– поддержка распределенных ограничений целостности.

Фрагментация отношений

Схема фрагментации отношения должна удовлетворять трем условиям:

Полнота: если отношение R разбивается на фрагменты R1, R2,…, Rn, то

URi = R

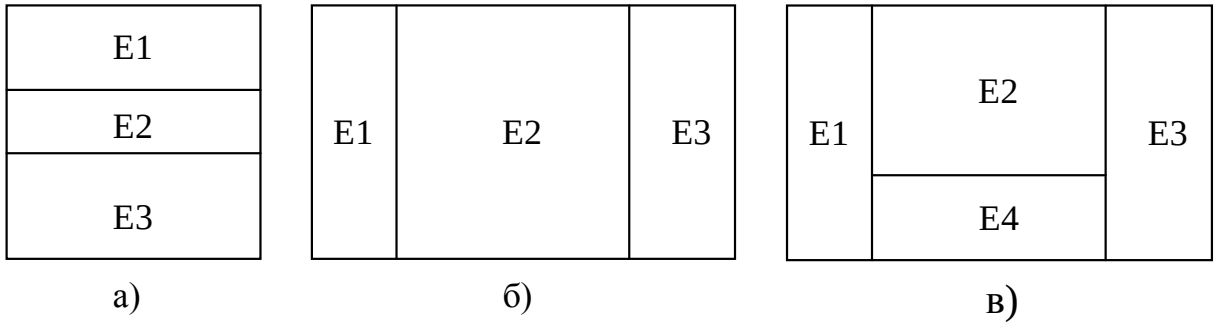
(Каждый кортеж должен входить хотя бы в один фрагмент).

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 отношений

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).

Фрагментация отношений

Целесообразность использования **горизонтальной фрагментации** может быть вполне очевидна, а может требовать выполнения детального анализа приложений. Анализ должен включать проверку предикатов (или условий) поиска, используемых в транзакциях или запросах, выполняемых в приложении. Для каждого из используемых атрибутов предикат может содержать единственное значение или несколько значений.

Стратегия определения типа фрагментации предполагает поиск набора минимальных (т.е. полных и релевантных) предикатов, которые можно использовать как основу для создания схемы фрагментации.

Набор предикатов является **полным** тогда и только тогда, когда вероятность обращения к любым двум строкам одного и того же фрагмента со стороны любой транзакции одинакова.

Предикат является **релевантным**, если существует, по крайней мере, одна транзакция, которая по-разному обращается к выделенным с помощью этого предиката фрагментам.

При **вертикальной фрагментации** в различные фрагменты объединяются атрибуты, используемые отдельными транзакциями.

Фрагментация в System R*

На начальной стадии создания этого проекта предполагалось иметь в системе средства горизонтального и вертикального разделения отношений распределенной базы данных. 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>

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>

Проблемы:

- поддержание согласованности разделов (распределенные ОЦ и т.д.):
- усложнение оптимизатора запросов;
- снижение производительности.

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 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**