Лабораторная работа № 1

Создание распределённой базы данных

Цель работы

Получить навыки проектирования и создания распределённой базы данных, выбора оптимального размещения таблиц базы данных по узлам.

Создание распределённой базы данных

В данной лабораторной работе нужно спроектировать распределённую базу данных, которая должна быть размещена на двух узлах (серверах баз данных). В качестве предметной области, для которой будет создаваться распределённая база данных, можно выбрать ту же предметную область, для которой на IV курсе в рамках курсового проекта создавалась база данных. В данной лабораторной работе нужно создать два варианта распределённой базы данных: один – при помощи распределения таблиц по узлам на основе некоторой логики, другой – при помощи оптимального распределения, полученного в результате решения оптимизационной задачи. Каждый вариант разбиения предполагает создание двух баз данных на разных узлах (в данной работе базы данных на разных узлах будут моделироваться при помощи схем).

Для размещения таблиц распределённой базы данных следует использовать базы данных **students51** и **students52**. Обе базы данных расположены на сервере **students.ami.nstu.ru**.

Для переноса курсового проекта, выполненного на IV курсе, из базы данных **students** необходимо с помощью программы PuTTY зайти на сервер **students.ami.nstu.ru** под бригадным логином (не текущего семестра, а под тем, с помощью которого ранее создавалась база данных) и выполнить описанные далее действия.

1. Экспортировать ранее созданную базу данных в SQL-файл. Для этого можно воспользоваться двумя способами:

   1) выполнить команду следующего вида:

```
pg_dump -U логин --encoding=utf-8 -f файл.sql -n имя_схемы --no-owner -x students
```

   Например, если курсовой проект находился в схеме **pm6503** и бригадный логин также был **pm6503**, команда будет выглядеть так:

```
pg_dump -U pm6503 --encoding=utf-8 -f pm6503_dump.sql -n pm6503 --no-owner -x students
```

   2) зайти в **phpPgAdmin** под групповым логином (**http://ami.nstu.ru/phpPgAdmin**, на странице выбрать сервер **students.ami.nstu.ru**), найти ранее созданную схему в базе **students**, выбрать её и в правом верхнем углу нажать **Экспорт**.

   После этого нужно поставить галочки так, как показано на рисунке 1:

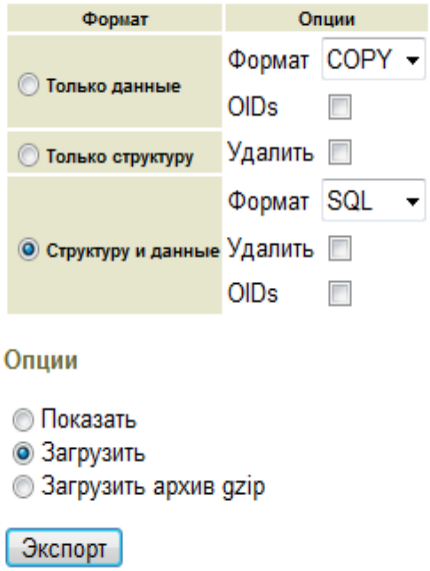

Рисунок 1 – Настройка экспорта схемы из базы данных

2. Для каждого из вариантов разбиений создать по две пары схем в базах **students51** и **students52**. Для этого можно использовать следующие команды:

```
new_schema имя_первой_схемы students51
new_schema имя_второй_схемы students52
new_schema имя_третьей_схемы students51
new_schema имя_четвертой_схемы students52
```

3. Используя sql-запросы, создающие таблицы, в каждую схему добавить необходимые таблицы. Это же можно сделать, загрузив таблицы из файла, содержащего sql-запросы на создание таблиц, командой вида

```
psql -U логин -d имя_базы_данных -f имя_sql_файла.sql
```

Выполнение запросов к распределённой базе данных

Распределённые запросы (т.е. те запросы, которые одновременно обращаются к нескольким базам данных) из базы **students51** реализуются следующим образом (с использованием библиотеки **dblink**):

```
dblink(text connstr, text sql),
```

где **sql** – sql-выражение, запускаемое на выполнение на удалённом узле (например, "**select * from customer**"), а **connstr** – строка соединения вида

```
"dbname=<имя_базы> user=<логин> password=<пароль>"
```

Примеры использования **dblink**:

```
select *
from   ttt.t1,
       public.dblink
         ('dbname=students52 user=login password=pswd',
          'select * from t2'
         ) as t2 (t21 text,
                  t22 text
                 )
where  t1.t12 = t2.t21;

select *
from   public.dblink
         ('dbname=students52 user=login password=pswd',
          'select id_tovar, name, id_type_tovar from имя_схемы.tovar'
         ) as t1 (id_tovar varchar,
                  name varchar,
                  id_type_tovar varchar
                 );
```

Для корректного измерения времени выполнения каждый запрос нужно выполнить несколько раз и взять среднее время выполнения.

Использование представлений

Представление (*view*) – объект данных, который не содержит никаких данных его владельца. Это тип таблицы, чьё содержание выбирается из других таблиц с помощью выполнения запроса. Поскольку значения в этих таблицах меняются, их значения автоматически будут показываться представлением.

Представление – это фактически запрос, который выполняется всякий раз, когда происходит обращение к представлению. Результат запроса при этом в каждый момент становится содержанием представления.

Для повышения производительности распределённых СУБД зачастую используют механизм репликации данных. Для учебных целей этот механизм можно реализовать с помощью кэширования удалённых таблиц СУБД в локальные представления. Таким образом, можно кэшировать удалённую таблицу в локальном представлении и обращаться не к удалённой таблице, а к её локальной копии, не используя **`dblink`**.

Представление можно создать следующим образом:

```
create view имя_схемы.tovar as
  select *
  from   public.dblink
           ('dbname=students52 user=login password=password',
            'select id_tovar, name, id_type_tovar from имя_схемы.tovar'
           ) as t1 (id_tovar varchar,
                    name varchar,
                    id_type_tovar varchar
                   );
```

В дальнейшем в качестве таблицы в запросах следует использовать **`имя_схемы.tovar`**.

Поиск оптимального размещения таблиц по узлам

Для поиска оптимального размещения таблиц по узлам нужно решить задачу линейного программирования, решением которой будет указание, какие таблицы следует разместить на первом узле, а какие – на втором.

Пусть:

$n = 2$ – число узлов в сети;

k – число таблиц базы данных;

m – общее число запросов к базе данных;

K_j, $j = \overline{1,n}$ – j-й узел в сети;

T_l, $l = \overline{1,k}$ – l-я таблица;

$L_l = e_l \cdot N_l$, $l = \overline{1,k}$ – объём l-й таблицы, где e_l – максимальный размер одной строки данных таблицы T_l (в байтах), N_l – число строк l-й таблицы;

b_j, $j = \overline{1,n}$ – объём памяти узла K_j для размещения таблиц;

Z_i, $i = \overline{1,m}$ – i-й запрос;

α_{il} – объём пересылаемых данных из таблицы T_l при выполнении i-го запроса. Определим $\alpha_{il} = d_{il} \cdot N_l$, где d_{il} – размер одной строки данных (в байтах) таблицы T_l, пересылаемой при выполнении i-го запроса, N_l – число строк l-й таблицы;

λ_{ij} – интенсивность появления i-го запроса, инициированного из узла K_j. λ_{ij} определяется произвольно на основе экспертной оценки пользователя и показывает, как часто запрос i будет вызываться из узла K_j;

$x_{ij} = \begin{cases} 1, & \text{если таблица } T_l \text{ в узле } K_j, \\ 0, & \text{иначе} \end{cases}$;

$\nu_{ij} = \sum_{l=1}^{k} \left(\alpha_{il} \cdot \left(1 - x_{lj}\right)\right)$ – интенсивность пересылаемых данных при выполнении i-го запроса;

$S = \sum_{i=1}^{m} \sum_{j=1}^{n} \left(\lambda_{ij} \cdot \nu_{ij} \right)$ – целевая функция, по которой нужно проводить оптимизацию.

Оптимизационную задачу нужно решать при следующих ограничениях:

$\sum_{j=1}^{n} x_{lj} = 1 \quad \forall l = \overline{1,k}$ – каждая таблица может находиться только в одном узле;

$\sum_{l=1}^{k} \left(L_l \cdot x_{lj} \right) \le b_j \quad \forall j = \overline{1,n}$ – суммарный объём таблиц в узле не должен превышать объём узла;

$x_{ij} \in \{0,1\} \quad \forall i = \overline{1,k} \quad \forall j = \overline{1,n}$.

Целевую функцию следует преобразовать следующим образом:

$$S = \sum_{i=1}^{m} \sum_{j=1}^{n} \left(\lambda_{ij} \cdot \nu_{ij} \right) = \sum_{i=1}^{m} \sum_{j=1}^{n} \left(\lambda_{ij} \cdot \sum_{l=1}^{k} \left(\alpha_{il} \cdot \left(1 - x_{lj}\right) \right) \right) = \sum_{i=1}^{m} \sum_{l=1}^{k} \left(\left(1 - x_{lj}\right) \cdot \sum_{j=1}^{n} \left(\lambda_{ij} \cdot \alpha_{il} \right) \right) =$$

$$= \sum_{i=1}^{m} \sum_{l=1}^{k} \sum_{j=1}^{n} \left(\lambda_{ij} \cdot \alpha_{il} \right) - \sum_{i=1}^{m} \sum_{l=1}^{k} \left(x_{li} \cdot \sum_{j=1}^{n} \left(\lambda_{ij} \cdot \alpha_{il} \right) \right)$$

Первая компонента в последнем равенстве – константа, которая может быть опущена, т.к. требуется найти только значения переменных x_{ij}.

Для базы данных необходимо составить следующие таблицы значений (таблицы 1, 2, 3 и 4):

Таблица 1 – Значения d_{il}

Z_i \ T_l	1	…	k
1			
…			
m			

Таблица 2 – Значения λ_{ij}

Z_i \ K_j	1	…	n
1			
…			
m			

Таблица 3 – Значения N_l

l	1	…	k
N_l			

Таблица 4 – Значения e_l

l	1	…	k
e_l			

Задание

I. На основе базы данных, разработанной в рамках курса «Базы данных», создать распределённую базу (база данных должна состоять не менее, чем из 8 таблиц) в двух вариантах:
   1. Разделить таблицы БД на две группы по какому-либо смысловому признаку. Например, если имеется база данных «Спортивный клуб», то в первую группу можно отнести таблицы, используемые для административной работы: список спортсменов, список тренеров, список баз и т.д., а во вторую группу – таблицы для соревнований: график соревнований, участие спортсменов в соревнованиях и т.д.
   2. Провести оптимальное размещение таблиц базы по двум узлам. Задачу линейного программирования можно решать любым подходящим алгоритмом с использованием любого существующего ПО. Оптимальное размещение следует рассчитывать, исходя из того, что суммарный объём узлов ($b_1 + b_2$) должен быть примерно равен $1.5 \cdot B$, где B – суммарный объём всех таблиц (это требование нужно, чтобы исключить ситуацию, когда решением оптимизационной задачи является размещение всех таблиц на одном узле).

II. Реализовать генератор данных больших объёмов. С помощью этого генератора заполнить оба варианта разбиения баз данных большим числом данных. В базе данных должны быть таблицы, имеющие несколько сотен записей, и таблицы, имеющие несколько тысяч записей. При этом должна сохраняться целостность как локальных баз данных, так и глобальной распределённой базы.

III. Оценить эффективность каждого варианта по набору sql-запросов, имевшемуся в задании к ранее выполненному курсовому проекту. Посчитать время выполнения каждого из запросов к базе данных и суммарное время работы по двум базам для каждого разбиения и сделать выводы.

Требования к оформлению отчёта

Отчёт по лабораторной работе должен включать в себя:

- титульный лист;
- описание использованной базы данных с точки зрения предметной области;
- структуру базы данных на языке SQL с указанием имён и типов полей всех таблиц;
- ER-диаграмму, на которой должны быть показаны все связи между таблицами базы и показаны графически два варианта разбиения базы;
- количество записей в каждой таблице, сгенерированных генератором;
- SQL-запросы для обоих вариантов разбиения баз данных;
- среднее время выполнения запросов (в мс) для обоих разбиений базы данных для каждого запроса и суммарное время выполнения запросов на обоих разбиениях;
- вывод, объясняющий, почему время выполнения запросов в одном из двух разбиений оказалось меньшим (большим, равным), чем в другом.

Отчёт не должен содержать орфографических, пунктуационных и смысловых ошибок. Все его разделы должны быть выдержаны в едином стиле оформления.

Замечание: текст генератора данных в отчёте можно не приводить.

Критерии оценивания качества работы

1. Полнота набора SQL-запросов для оценки эффективности:
   1 – не менее 7 запросов, из которых не менее 5 распределённых;
   0 – не менее 3 запросов, из которых не менее 2 распределённых;
   л.р. не принимается – менее 3 запросов или менее 2 распределённых.
2. Наличие ER-диаграммы с изображёнными разбиениями таблиц по узлам:
   1 – есть;
   0 – нет.
3. Содержание отчёта:
   1 – отчёт удовлетворяет всем требованиям;
   0 – отчёт удовлетворяет не всем требованиям.
4. Глубина понимания материала лабораторной работы каждым членом бригады:
   1 – быстрые и правильные ответы на все вопросы;
   0 – не на все вопросы ответы правильные и быстрые;
   л.р. не принимается – на половину вопросов ответы неправильные.