Санкт-Петербургский Политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчёт о лабораторной работе №1

Дисциплина: Базы данных

Тема: Оптимизация SQL-запросов

Выполнил студент гр. 43501/3 _____ В.Е. Бушин
(подпись)

Руководитель _____ А.В. Мяснов
(подпись)

“_” _____2016 г.

Санкт-Петербург

2016

Цель работы
Получить практические навыки создания эффективных SQL-запросов.

Программа работы

1. Ознакомьтесь со способами профилирования и интерпретации планов выполнения SQL-запросов
2. Ознакомьтесь со способами оптимизации SQL-запросов с использованием:
    - индексов
    - модификации запроса
    - создания собственного плана запроса
    - денормализации БД
3. Нагенерируйте данные во всех таблицах, если это ещё не сделано
4. Выберите один из существующих или получите у преподавателя новый "тяжёлый" запрос к Вашей БД
5. Оцените производительность запроса и проанализируйте результаты профилирования (для этого используйте SQL Editor в средстве IBExpert)
6. Выполните оптимизацию запроса двумя или более из указанных способов, сравните полученные результаты
7. Продемонстрируйте результаты преподавателю
8. Напишите отчёт с подробным описанием всех этапов оптимизации и выложите его в Subversion

Общие сведения

Индекс — объект базы данных, создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путем последовательного просмотра таблицы строка за строкой может занимать много времени. Индекс формируется из значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы и, таким образом, позволяет искать строки, удовлетворяющие критерию поиска. Ускорение работы с использованием индексов достигается в первую очередь за счёт того, что индекс имеет структуру, оптимизированную под поиск — например, сбалансированного дерева.

Модификация запросов
Явное указание порядка обхода таблиц с помощью конструкций JOIN.

План выполнения запроса — последовательность операций, необходимых для получения результата SQL-запроса в реляционной СУБД.
Перед выполнением запроса происходит его подготовка – составление плана выполнения запроса (последовательности обхода и соединения таблиц).
Операции извлечения данных:

- NATURAL – полный перебор, до тех пор пока не найдет требуемые данные, допустим при извлечении всех данных таблицы;
- INDEX(<index1>,…) – поиск по индексу, обычно более эффективно, используется при соединениях и вычислении условий;
- ORDER <index> - полный перебор с упорядочиванием по заданному индексу, можно использовать при order by и group by.
- JOIN (<select1>,<select2>,…) – соединение двух или более потоков в один, осуществляется перевод всех записей <select1> и поиск для них записей

<select2> и т.д., эффективное слияние при наличии индексов;

- MERGE (<select1>,<select2>,…) – выбирает и сортирует сразу все потоки и производит слияние за один проход, эффективен при отсутствии индексов
- SORT(<select>) – сортировка потока.

При автоматическом создании планов используется статистика по индексам. Возможно явное указание плана в запросе – перед ORDER BY.

Денормализация — намеренное приведение структуры базы данных в состояние, не соответствующее критериям нормализации, обычно проводимое с целью ускорения операций чтения из базы за счет добавления избыточных данных.

Выполнение работы:

Перед выполнением задания база данных была преобразована следующим образом:

```
create table Season(
    id int primary key,
    begining date,
    ending date
);

create table Match(
    id int primary key,
    matchDate DATE,
    home int not null,
    away int not null,
    home_scored int,
    away_scored int,
    seasonID int not null
);

alter table Match add constraint match_to_home
    foreign key (home) references Club(id);
alter table Match add constraint match_to_away
    foreign key (away) references Club(id);
alter table Match add constraint match_to_season
    foreign key (seasonID) references Season(id);

create domain StageDomain
    AS integer check (value in (1,2,4,8,16,32));

create table Match_champ(
    id int primary key,
    tour int,
    matchID int not null
);

alter table Match_champ add constraint match_champ_to_match
    foreign key (matchID) references Match(id);

create table Match_cup(
    id int primary key,
    stage StageDomain,
    winner int not null,
    matchID int not null
);

alter table Match_cup add constraint match_cup_to_winner
```

```
    foreign key (winner) references Club(id);
alter table Match_cup add constraint match_cup_to_match
    foreign key (matchID) references Match(id);
```

В итоге база данных стала выглядеть следующим образом:

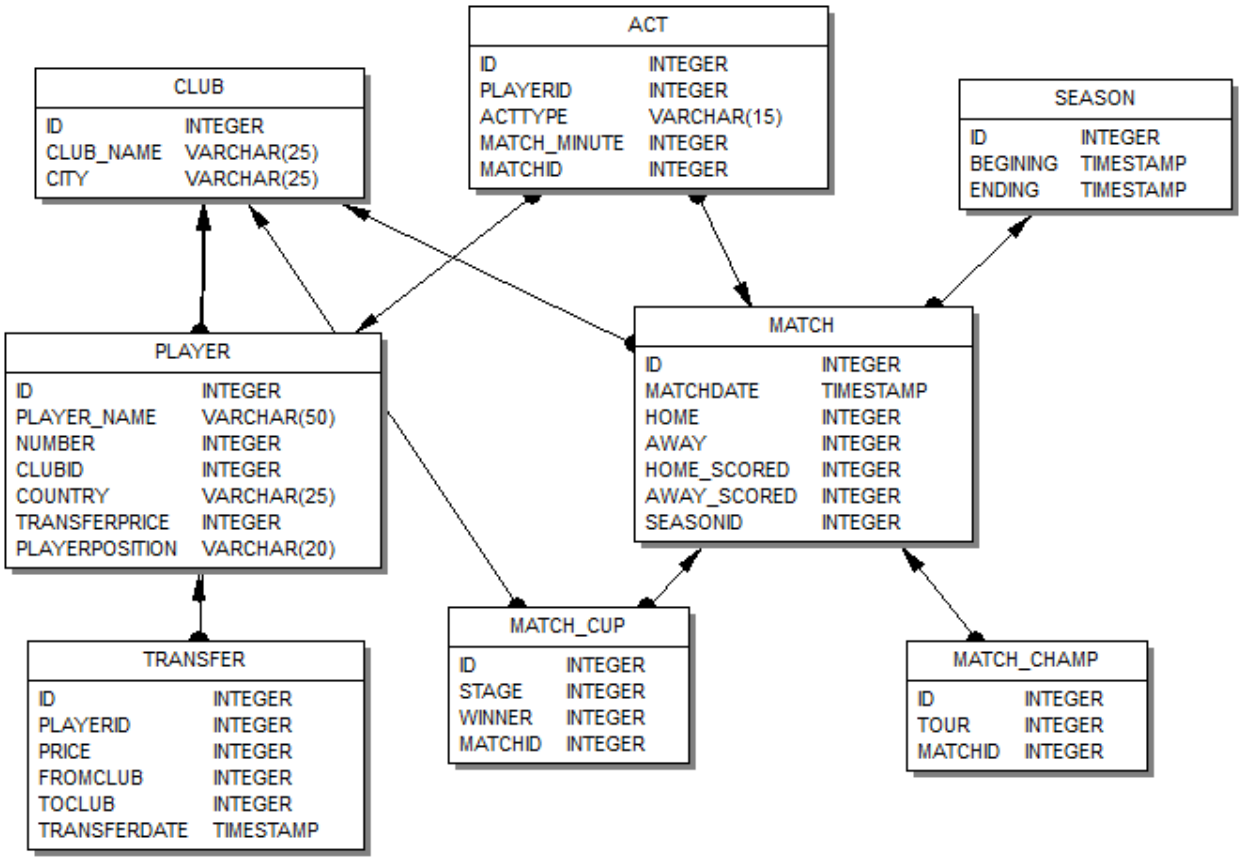

Рис.1. Диаграмма базы данных

Для выполнения работы был выбран следующий запрос – вывод за всё время трансферной стоимости игрока и его эффективности (действия игрока в матче):

```
select Player.player_name as name, Transfer.price as price,
    count(Act.acttype) as efficiency from Player, Transfer, Act
    where Player.id=Act.playerid and Player.id=Transfer.playerid
    group by Player.player_name, Act.acttype, Transfer.price;
```

В таблицах сгенерировано большое количество данных(10000, а в таблице Act 30000). Для оценки скорости выполнения запроса он был выполнен несколько раз, получены следующие времена выполнения: 203ms, 204ms, 188ms, 172 ms, 219ms. Средняя скорость выполнения запроса 197,2ms.

Добавлены индексы:

```
create index player_id on player(id);
create index price on Transfer(price);
create index stat on Act(acttype,playerid);
```

Время выполнения запроса было проверено на тех же наборах данных: 187ms, 188ms, 266ms, 203ms, 187ms. Среднее время выполнения запроса теперь: 206ms.

Среднее время выполнения запроса практически не изменилось (увеличилось примерно на 5%). Исходя из этого, можно сделать вывод, что в данном случае индексы не помогли в оптимизации.

Добавлен план выполнения запроса:

```
plan join(player natural, Act natural, transfer natural)
```

Время выполнения запроса было проверено на тех же наборах данных: 156ms, 109ms, 125ms, 141ms, 140ms. Среднее время выполнения запроса теперь: 134ms.
Среднее время выполнения запроса уменьшилось на 32%, что говорит об эффективности применения плана выполнения запроса в данном случае.

Денормализация базы данных:

В таблицу Player добавлено поле со значением трансферной стоимости игрока – transferprice, таким образом из запроса исключается таблица Transfer и сам запрос немного изменяется:

```
select Player.player_name as name, Player.transferprice as price,
    count(Act.acttype) as efficiency from Player, Act
    where Player.id=Act.playerid group by Player.player_name, Act.acttype,
Player.transferprice;
```

Время выполнения запроса было проверено на тех же наборах данных: 203ms, 204ms, 188ms, 218ms, 235ms. Среднее время выполнения запроса теперь: 209ms. Среднее время выполнения запроса практически не изменилось (увеличилось примерно на 5%).
Попробуем добавить ещё и план выполнения запроса, чтобы уменьшить время выолнения запроса:

```
plan join(player natural, Act natural)
```

Время выполнения запроса было проверено на тех же наборах данных: 110ms, 109ms, 94ms, 105ms, 101ms. Среднее время выполнения запроса теперь: 104ms. Среднее время выполнения запроса уменьшилось на 47%, что говорит об эффективности применения плана выполнения запроса и денормализации базы данных в данном случае.

Вывод

В ходе выполнения работы была проведена оптимизация запросами разными способами: с помощью добавления индексов, плана запроса и денормализации БД. Не все из этих способов оказались удачными: при добавлении индексов и при денормализации БД время выполнения запроса немного увеличилось. Но в свою очередь при создании собственного плана запроса и также совмещении плана запроса с денормализацией БД время выполнения запроса значительно уменьшилось. Отсюда можно сделать следующий вывод: самый эффективный способ оптимизации запроса – это совмещение нескольких способов уменьшения время выполнения запроса.