Санкт-Петербургский политехнический университет
Кафедра компьютерных систем и программных технологий

Отчёт по вводной лабораторной работе

Дисциплина: Базы данных

Тема: Получить практические навыки создания эффективных SQL-запросов.

Выполнил студент гр. 43501/4 _____ Чеботарёв Г.М.
(подпись)
Руководитель _____ Мяснов А.В.
(подпись)

Санкт -Петербург
2016

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 Subversio

Ход работы:

1. *Сделать вывод по клубам за некоторый период: среднее место члена клуба и суммарная стоимость активов.* Мест у членов клуба нет, будем искать среднее кол-во побед: «*Сделать вывод по клубам: среднее число побед членов клуба, а также актив для каждого клуба, рассчитать от заданной даты».*

Первая реализация:

```
SELECT T1.id_chess_club, T3.name, T1.wins, T2.activ, T2.date_purchase
FROM
(
# получение среднего числа побед
SELECT id_chess_club, AVG(count_of_wins) as wins FROM chess_players GROUP BY
chess_players.id_chess_club
) T1,
(
# получение суммы покупок по клубам
select id_club, SUM(cost) as activ from purchase_accounting  where
date_purchase > '2007-10-23' and date_purchase < '2007-10-26' GROUP BY
id_club
) T2,
(
# получение названия клуба
select id_chess_club, name from chess_club GROUP BY id_chess_club
) T3
WHERE T2.id_club = T1.id_chess_club and T3.id_chess_club = T1.id_chess_club;
```

Вторая реализация:

```
SELECT pa.id_club, cc.name, pa.date_purchase, SUM(pa.cost) as activ,
AVG(cp.count_of_wins) as wins
FROM chess_club as cc
INNER JOIN purchase_accounting as pa ON pa.id_club = cc.id_chess_club
INNER JOIN chess_players as cp on cp.id_chess_club = cc.id_chess_club
where date_purchase > '2007-10-23' and date_purchase < '2007-10-26' GROUP BY
id_club;
```

В результате выполнения получим:

id_chess_club	name	wins	activ	date_purchase
5	Большие Васюки5	91,0000	530	2007-10-25
11 860	Большие Васюки11860	28,0000	885	2007-10-24
18 416	Большие Васюки18416	143,0000	262	2007-10-25
28 859	Большие Васюки28859	406,0000	1 111	2007-10-24

Рис.1-результат выполнения.

Каждому клубу соответствует актив, приобретенный в выбранный период и среднее кол-во побед в клубе. Первая реализация выполняется в лучшем случае за 13 секунд, в то время как вторая выполняется практически моментально:

```
SELECT T1.id_chess_club, T3.name, T1.wins, T2.activ, T2.date_purchase
FROM
(
# получение среднего числа побед
SELECT id_chess_club, AVG(count_of_wins) as wins FROM chess_players GROUP BY
chess_players.id_chess_club
) T1,
(
# получение суммы покупок по клубам
select id_club, SUM(cost) as activ from purchase_accounting  where
date_purchase  >  '2007-10-23'  and  date_purchase  <  '2007-10-26'  GROUP  BY
id_club
) T2,
(
# получение названия клуба
select id_chess_club, name from chess_club GROUP BY id_chess_club
) T3
WHERE T2.id_club = T1.id_chess_club and T3.id_chess_club = T1.id_chess_club;
/* Affected rows: 0  Найденные строки: 4  Предупреждения: 0  Длительность  1
query: 13,165 sec. */
```

```
SELECT pa.id_club, cc.name, pa.date_purchase, SUM(pa.cost) as activ,
AVG(cp.count_of_wins) as wins
FROM chess_club as cc
INNER JOIN purchase_accounting as pa ON pa.id_club = cc.id_chess_club
INNER JOIN chess_players as cp on cp.id_chess_club = cc.id_chess_club
where date_purchase > '2007-10-23' and date_purchase < '2007-10-26' GROUP BY
id_club;
/* Affected rows: 0  Найденные строки: 4  Предупреждения: 0  Длительность  1
query: 0,735 sec. */
```

Третий вид оптимизации - индексирование БД. **Индексы** - это специальные структуры в базах данных, которые позволяют ускорить поиск и сортировку по определенному полю или набору полей в таблице, а также используются для обеспечения уникальности данных. Для того, чтобы повысить скорость поиска данных по таблице, создадим индексы «ключи». Создание индекса происходит при помощи специальной команды:

```
CREATE INDEX age ON chess_players(age);
```

После этой операции MySQL начнет использовать созданный индекс для повышения эффективности поисковых запросов по конкретной колонке. Создадим индексы для всех нужных полей и повторим последний запрос:

```
SELECT pa.id_club, cc.name, pa.date_purchase, SUM(pa.cost) as activ,
AVG(cp.count_of_wins) as wins
FROM chess_club as cc
INNER JOIN purchase_accounting as pa ON pa.id_club = cc.id_chess_club
INNER JOIN chess_players as cp on cp.id_chess_club = cc.id_chess_club
where date_purchase > '2007-10-23' and date_purchase < '2007-10-26' GROUP BY
id_club;
/* Affected rows: 0  Найденные строки: 4  Предупреждения: 0  Длительность  1
query: 0,094 sec. */
```

Время запроса составило 0,094 секунды, что в восемь раз быстрее запроса по неиндексированной БД. После оптимизации запроса и индексирования базы данных время запроса уменьшилось в 160 раз.

2. *Рассчитать доходы и расходы каждого клуба за некоторый период.*
Оптимизированная версия запроса:

```
SELECT pa.id_club,cc.name,SUM(pa.cost) as costs,t.cost as
income,pa.date_purchase
FROM chess_club as cc
INNER JOIN purchase_accounting as pa ON pa.id_club = cc.id_chess_club
INNER JOIN transfer as t on t.id_club = cc.id_chess_club
where pa.date_purchase > '2015-09-21' and pa.date_purchase < '2015-11-30' and
t.date_transfer > '2015-09-21' and t.date_transfer < '2015-11-30' GROUP BY
id_club;
```

В результате выполнения выведется

id_club	name	costs	income	date_purchase
1	Большие Васюки1	2 535	1 100	2015-11-20
43	Большие Васюки43	3 136	112 208	2015-09-22
44	Большие Васюки44	1 250	412 432	2015-09-22
57	Большие Васюки57	1 428	474 669	2015-09-22
58	Большие Васюки58	885	284 892	2015-09-22
59	Большие Васюки59	343	95 116	2015-09-22
68	Большие Васюки68	748	236 682	2015-09-22
69	Большие Васюки69	205	46 905	2015-09-22
76	Большие Васюки76	294	78 024	2015-09-22

Рис.2-результат выполнения.

Время запроса по НЕ индексированной базе составило порядка 2 секунд:

```
SELECT pa.id_club,cc.name,SUM(pa.cost) as costs,t.cost as
income,pa.date_purchase
FROM chess_club as cc
INNER JOIN purchase_accounting as pa ON pa.id_club = cc.id_chess_club
INNER JOIN transfer as t on t.id_club = cc.id_chess_club
where pa.date_purchase > '2015-09-21' and pa.date_purchase < '2015-11-30' and
t.date_transfer > '2015-09-21' and t.date_transfer < '2015-11-30' GROUP BY
id_club;
```

```
/* Affected rows: 0  Найденные строки: 9  Предупреждения: 0  Длительность  1
query: 1,604 sec. */
```

Для индексированной БД:

```
/* Affected rows: 0  Найденные строки: 9  Предупреждения: 0  Длительность  1
query: 0,203 sec. */
```

В результате индексирования БД скорость выполнения запроса возросла в 8 раз.
Все запросы проводились на заполненной данными БД. Количество записей в таблицах колебалось от 40 до 600 тыс.

Вывод:

Одной из важнейших характеристик современных информационных систем является реактивность. Базы данных неотъемлемая часть таких систем, как следствие, к БД предъявляются высокие требования. Для того, чтобы соответствовать поставленной планке, необходимо выдавать максимальное быстродействие БД. Этого можно добиться оптимизацией. В выполненной л.р. были продемонстрированы два из четырех основных способа оптимизация: оптимизация запроса и индексирование БД. Скорость работы при использовании обоих методов выросла на два порядка в среднем.

1) индексирование лучше применять в таблицах, которые обновляются не ежесекундно, т.к. добавление в индексированную таблицу происходит значительно медленнее чем в неиндексированную.

2) Оптимизированность запросы должна быть выполнено всегда без исключения. Любой запрос должен быть написан наиболее оптимальным образом, для получения быстродействия.

3) Деморализация БД – данный метод как раз поможет в тех случаях, когда запрос цепляет сразу несколько таблиц, и подобные запросы происходят достаточно регулярно. В таком случае, дублирование и объединение этих таблиц в одну (или некоторых полей) отличный вариант для повышения эффективности.

4) Создание собственного плана запроса – так же позволяется повысить эффективность выполнения запроса. В неиндексированной БД движок mysql проверяет всю таблицу подряд. В случае если для поля, по которому идет поиск, создан индекс, то поиск выполняется согласно индексу. Более того, с помощью специальных функций план может быть изменен, например, используя STRAIGHT_JOIN вместо JOIN, можно явно указать с какой таблицы начинать поиск (или любую другую операцию). Иногда такое явное указание может дать более эффектный результат.

В работе применены первый и второй метод, т.к. нужды в 3 и 4 в данной БД нет.