Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский политехнический университет Петра Великого»
Институт компьютерных наук и технологий
Кафедра Компьютерных систем и программных технологий

Отчёт по лабораторной №8

Дисциплина: Базы данных

Тема: Оптимизация SQL-запросов

Выполнил студент гр. 43501/4 _____ В.С. Филиппов
(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

Выполнение работы

1. Планы запросов

Перед выполнением запроса комплект программ подготовки - известный как оптимизатор начинает анализировать столбцы и операции запроса для вычисления самого быстрого способа выполнения. Подготовка начинается с просмотра индексов таблицы и используемых столбцов. Работая таким образом с последовательностью путей решения (каждый из которых имеет свою "стоимость"), оптимизатор создает план - некий вид "дорожной карты" того пути, по которому сервер будет следовать при выполнении запроса. Конечный план выбирается по критерию "самой дешевой" дороги, оцениваемой в соответствии с индексами, которые могут быть использованы.

В СУБД Firebird для рабты с планами имеются следующие инструменты isql:

* SET PLAN ON для отображения плана в самом начале вывода запроса SELECT.

* SET PLANONLY для рассмотрения запроса и просмотра плана без фактического выполнения запроса (любого, не только запросов SELECT).

Также можно перекрыть план запроса оптимизатора вашим собственным планом, включив предложение PLAN в оператор запроса.

2. Способы оптимизации запросов с использованием:

а) индексов

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.

В СУБД Firebird имеются ASC и DESC индексы для поиска в прямом и обратном порядке. Например, DESC лучше использовать для следующего условия в запросе **where email similar to '%@ya.ru'**, иначе поиск будет осуществлятся полным перебором. Так же обратный индекс может ускорить работу функции **max()**.

Чаще для выигрыша по времени всего индексы применяют:

- ✓ к столбцам, указанным в предложениях ORDER BY или GROUP BY
- ✓ при слиянии потоков данных на основании соответствия значений, явно или неявно указанных в критерии ON
- ✓ определения, является ли его значение больше, равно или меньше значения константы

Недостатки индексов:

- ☝ увеличение числа индексов замедляет операции добавления, обновления, удаления строк таблицы, поскольку при этом приходится обновлять сами индексы
- ☝ индексы занимают дополнительный объем памяти, поэтому перед созданием индекса следует убедиться, что планируемый выигрыш в производительности запросов превысит дополнительную затрату ресурсов компьютера на сопровождение индекса

В СУБД Firebird при определении ваших собственных индексов крайне важно исключить создание любых индексов, которые дублируют индексы, в том числе

и автоматически сгенерированные индексы. Это ставит оптимизатор в печальную ситуацию выбора между равными индексами. В большинстве случаев он разрешит проблему, не выбрав ни одного.

б) денормализации БД

Иногда в SELECT запросе данные вычисляются, или берутся из другой таблицы. Чтобы не тратить время на вычисление, или на слияния с другой таблицей, часто применяют денормализацию. В этом случае добавляют столбцы, в одну таблицу, и триггер, чтобы этот столбец генерировался по событию, чтобы обращения в другую таблицу, или вычисление данных было произведено по триггеру, а при SELECT запросе их останется взять из новых столбцов.

3. Генерация данных во все таблицы

Ранее было сгенерированы тестовые данные для таблиц. Их количество приведено на рис. 1.

TABLE_NAME	COUNT_
events	109 143
matches	100 851
players	100 517
clubs	100 503
current_club_players_list	100 409
seasons	26 039
leagues	20 007
club_players_list_history	507
standings	3

Рис.1. Количество записей в таблицах

4. Выбор «тяжёлого» запроса для оптимизации

Одним из самых медленных по времени выполнения запросов в нашей базе данных это был **selectTopClubs** – который выводит команд с наибольшим количеством побед за выбранный период. Его текст представлен в листинге 1.

Листинг 1. Неоптимизированный запрос.

```
create or alter view selectTopClubs as
    select clubs.name, wins from
        (
        select first 5 cid, count(cid) wins from
            (
            select first_club_id cid from matches
                where matches.match_date between '01-SEP-2015' and '31-DEC-
2015'
                  and matches.first_club_goals > matches.second_club_goals
            union all
            select second_club_id cid from matches
                where matches.match_date between '01-SEP-2015' and '31-DEC-
2015'
                  and matches.first_club_goals < matches.second_club_goals
            )
```

```
        group by cid order by wins desc
        )
        , clubs
    where clubs.club_id = cid;
```

Результат его выполнения приведён на рис. 2.

NAME	WINS
ppraijeqafq	2
Zenit	1
Spartak	1
jxcjkwwkrdzn	1
ybydmihvk	1

Рис.2. результат выполнения неоптимизированного запроса.

5. Оценка производительности запроса и анализ результатов профилирования

Данные о выполнении неоптимизированного запроса, приведённые в IBExpert представлены в листинге 2.

Листинг 2. Производительность неоптимизированного запроса.

```
Plan
PLAN JOIN (SORT ((SELECTTOPCLUBS MATCHES NATURAL)
PLAN (SELECTTOPCLUBS MATCHES NATURAL)), SELECTTOPCLUBS CLUBS INDEX
(RDB$PRIMARY2))

Adapted Plan
PLAN JOIN (SORT ((SELECTTOPCLUBS MATCHES NATURAL)
PLAN (SELECTTOPCLUBS MATCHES NATURAL)), SELECTTOPCLUBS CLUBS INDEX
(INTEG_5))

------ Performance info ------
Prepare time = 0ms
Execute time = 297ms
Avg fetch time = 59,40 ms
Current memory = 9 925 616
Max memory = 10 138 240
Memory buffers = 2 048
Reads from disk to cache = 0
Writes from cache to disk = 0
Fetches from cache = 405 882
```

6. Оптимизации запросов с использованием:

а) индексов

Для запроса был создан индекс в таблице matches по столбцу match_date

Листинг 3. Добавление индекса по дате.

```
CREATE ASC
INDEX matches_match_date ON matches(match_date);
```

Проверим увеличилась ли скорость запроса.

Листинг 4. Производительность после добавления индекса `matches_match_date`.

```
Plan
PLAN JOIN (SORT ((SELECTTOPCLUBS MATCHES INDEX (MATCHES_MATCH_DATE))
```

```
PLAN (SELECTTOPCLUBS MATCHES INDEX (MATCHES_MATCH_DATE))), SELECTTOPCLUBS
CLUBS INDEX (RDB$PRIMARY2))

Adapted Plan
PLAN JOIN (SORT ((SELECTTOPCLUBS MATCHES INDEX (MATCHES_MATCH_DATE))
PLAN (SELECTTOPCLUBS MATCHES INDEX (MATCHES_MATCH_DATE))), SELECTTOPCLUBS
CLUBS INDEX (INTEG_5))

------ Performance info ------
Prepare time = 0ms
Execute time = 47ms
Avg fetch time = 9,40 ms
Current memory = 9 931 152
Max memory = 14 171 664
Memory buffers = 2 048
Reads from disk to cache = 0
Writes from cache to disk = 0
Fetches from cache = 21 292
```

Видно, что в план запроса добавилась сортировка по созданному индексу, и быстродействие увеличилось примерно на 1 порядок.

Попробуем добавить ещё индексы по количеству забитых мячей 1ой и 2ой команды.

Листинг 5. Добавление индексов по голам.

```
CREATE ASC
INDEX matches_first_club_goals ON matches(first_club_goals);

CREATE ASC
INDEX matches_second_club_goals ON matches(second_club_goals);
```

Проверим увеличилась ли скорость запроса.

Листинг 6. Производительность после добавления индексов по голам.

```
Plan
PLAN JOIN (SORT ((SELECTTOPCLUBS MATCHES INDEX (MATCHES_MATCH_DATE))
PLAN (SELECTTOPCLUBS MATCHES INDEX (MATCHES_MATCH_DATE))), SELECTTOPCLUBS
CLUBS INDEX (RDB$PRIMARY2))

Adapted Plan
PLAN JOIN (SORT ((SELECTTOPCLUBS MATCHES INDEX (MATCHES_MATCH_DATE))
PLAN (SELECTTOPCLUBS MATCHES INDEX (MATCHES_MATCH_DATE))), SELECTTOPCLUBS
CLUBS INDEX (INTEG_5))

------ Performance info ------
Prepare time = 0ms
Execute time = 47ms
Avg fetch time = 9,40 ms
Current memory = 9 931 256
Max memory = 14 821 896
Memory buffers = 2 048
Reads from disk to cache = 0
Writes from cache to disk = 0
Fetches from cache = 21 292
```

Видно, что быстродействие не увеличилось. Это потому, что оптимизатор не включил эти индексы в план запроса. Действительно, в данном запросе довольно сложно включить их в план.

б) денормализации БД

Добавим в таблицу matches столбец с вычисленной разницей столбцов first_club_goals и second_club_goals, чтобы в дальнейшем производить поиск по уже заранее вычисленным значениям. Для этого на всякий случай скопируем нашу таблицу в новую таблицу.

Листинг 7. Добавление столбца разницы голов.

```
CREATE TABLE MATCHES_OPT (
    MATCH_ID                        INTEGER NOT NULL,
    MATCH_DATE                      DATE,
    SEASON_ID                       INTEGER,
    FIRST_CLUB_ID                   INTEGER,
    SECOND_CLUB_ID                  INTEGER,
    FIRST_CLUB_PERCENT              SMALLINT,
    FIRST_CLUB_SHORTS               SMALLINT,
    SECOND_CLUB_SHORTS              SMALLINT,
    FIRST_CLUB_GOALS                SMALLINT,
    SECOND_CLUB_GOALS               SMALLINT,
    FIRST_CLUB_GOALS_MINUS_SECOND   SMALLINT
);

CREATE GENERATOR gen_match_id_opt;
SET GENERATOR gen_match_id_opt TO 0;
set term !! ;
CREATE TRIGGER MATCHES_OPT_BI FOR matches_opt
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
NEW.match_id = GEN_ID(gen_match_id_opt, 1);
END!!
set term ; !!

INSERT INTO matches_opt(
                            MATCH_DATE                      ,
                            SEASON_ID                       ,
                            FIRST_CLUB_ID                   ,
                            SECOND_CLUB_ID                  ,
                            FIRST_CLUB_PERCENT              ,
                            FIRST_CLUB_SHORTS               ,
                            SECOND_CLUB_SHORTS              ,
                            FIRST_CLUB_GOALS                ,
                            SECOND_CLUB_GOALS               ,
                            FIRST_CLUB_GOALS_MINUS_SECOND
                        )
SELECT
                            MATCH_DATE                      ,
                            SEASON_ID                       ,
                            FIRST_CLUB_ID                   ,
                            SECOND_CLUB_ID                  ,
                            FIRST_CLUB_PERCENT              ,
                            FIRST_CLUB_SHORTS               ,
                            SECOND_CLUB_SHORTS              ,
                            FIRST_CLUB_GOALS                ,
                            SECOND_CLUB_GOALS               ,
                            FIRST_CLUB_GOALS_MINUS_SECOND
FROM matches;
```

Далее заполним таблицу столбец вычисленными значениями.

Листинг 8. Заполнение столбца с разницей голов.

```
INSERT INTO matches(first_club_goals_minus_second)
SELECT first_club_goals - second_club_goals FROM matches;
```

Создадим запрос select_top_clubs для новой таблицы.

Листинг 9. Заполнение столбца с разницей голов.

```
CREATE OR ALTER VIEW SELECTTOPCLUBS_OPT(
    NAME,
    WINS)
AS
select clubs.name, wins from
        (
        select first 5 cid, count(cid) wins from
            (
            select first_club_id cid from matches_opt
                where matches_opt.match_date between '01-SEP-2015' and '31-
DEC-2015'
                  and matches_opt.first_club_goals_minus_second > 0
            union all
            select second_club_id cid from matches_opt
                where matches_opt.match_date between '01-SEP-2015' and '31-
DEC-2015'
                  and matches_opt.first_club_goals_minus_second < 0
            )
        group by cid order by wins desc
        )
        , clubs
    where clubs.club_id = cid;
```

Выполним запрос, и посмотрим скорость работы.

Листинг 10. Производительность запроса для денормализованной таблицы.

```
Plan
PLAN JOIN (SORT ((SELECTTOPCLUBS_OPT MATCHES_OPT NATURAL)
PLAN (SELECTTOPCLUBS_OPT MATCHES_OPT NATURAL)), SELECTTOPCLUBS_OPT CLUBS
INDEX (RDB$PRIMARY2))

Adapted Plan
PLAN JOIN (SORT ((SELECTTOPCLUBS_OPT MATCHES_OPT NATURAL)
PLAN (SELECTTOPCLUBS_OPT MATCHES_OPT NATURAL)), SELECTTOPCLUBS_OPT CLUBS
INDEX (INTEG_5))

------ Performance info ------
Prepare time = 15ms
Execute time = 297ms
Avg fetch time = 59,40 ms
Current memory = 10 062 048
Max memory = 15 071 200
Memory buffers = 2 048
Reads from disk to cache = 0
Writes from cache to disk = 0
Fetches from cache = 405 882
```

Видно, что по сравнению с нормализованной таблицей производительность не изменилась. Тогда попробуем так же добавить индексы.

Листинг 11. Добавление индекса по дате.

```
CREATE ASC
INDEX matches_match_date ON matches_opt(match_date);
```

Производительность такая же как у нормализованной таблицы с таким же индексом.

Листинг 12. Производительность.

```
Plan
PLAN JOIN (SORT ((SELECTTOPCLUBS_OPT MATCHES_OPT INDEX (MATCHES_MATCH_DATE))
PLAN (SELECTTOPCLUBS_OPT MATCHES_OPT INDEX (MATCHES_MATCH_DATE))),
SELECTTOPCLUBS_OPT CLUBS INDEX (RDB$PRIMARY2))

Adapted Plan
PLAN JOIN (SORT ((SELECTTOPCLUBS_OPT MATCHES_OPT INDEX (MATCHES_MATCH_DATE))
PLAN (SELECTTOPCLUBS_OPT MATCHES_OPT INDEX (MATCHES_MATCH_DATE))),
SELECTTOPCLUBS_OPT CLUBS INDEX (INTEG_5))

------ Performance info ------
Prepare time = 0ms
Execute time = 47ms
Avg fetch time = 9,40 ms
Current memory = 10 173 992
Max memory = 15 071 200
Memory buffers = 2 048
Reads from disk to cache = 0
Writes from cache to disk = 0
Fetches from cache = 21 068
```

Добавим индекс по разнице голов.

Листинг 13. Добавление индекса по новому столбцу.

```
CREATE ASC
INDEX first_club_goals_minus_sec_idx ON
matches_opt(first_club_goals_minus_second);
```

Проверим производительность.

Листинг 14. Производительность.

```
Plan
PLAN JOIN (SORT ((SELECTTOPCLUBS_OPT MATCHES_OPT INDEX (MATCHES_MATCH_DATE,
FIRST_CLUB_GOALS_MINUS_SEC_IDX))
PLAN (SELECTTOPCLUBS_OPT MATCHES_OPT INDEX (MATCHES_MATCH_DATE,
FIRST_CLUB_GOALS_MINUS_SEC_IDX))), SELECTTOPCLUBS_OPT CLUBS INDEX
(RDB$PRIMARY2))

Adapted Plan
PLAN JOIN (SORT ((SELECTTOPCLUBS_OPT MATCHES_OPT INDEX (MATCHES_MATCH_DATE,
FIRST_CLUB_GOALS_MINUS_SEC_IDX))
```

```
PLAN (SELECTTOPCLUBS_OPT MATCHES_OPT INDEX (MATCHES_MATCH_DATE,
FIRST_CLUB_GOALS_MINUS_SEC_IDX))), SELECTTOPCLUBS_OPT CLUBS INDEX (INTEG_5))

------ Performance info ------
Prepare time = 0ms
Execute time = 15ms
Avg fetch time = 3,00 ms
Current memory = 10 175 896
Max memory = 15 071 200
Memory buffers = 2 048
Reads from disk to cache = 0
Writes from cache to disk = 0
Fetches from cache = 283
```

Видно, что данный индекс оптимизатор включил в план запроса и производительность увеличилась ещё в 2 раза.

Вывод

Оптимизация с помощью индексов является наиболее простой и эффективной. Необходимо лишь создать индекс по столбцу, который используется в условиях поиска, и скорость выполнения запроса увеличивается в разы (~ в 10 раз в нашем случае).

Денормализация базы данных куда более сложная процедура, и не всегда может ускорить выполнение запроса, как это было в нашем случае. Однако удалось достигнуть прироста производительности после денормализации при помощи индексов. Для исходной таблицы созданные индексы по голам оптимизатор не смог добавить в план запроса. Для денормализованной таблицы был создан индекс для нового столбца, который оптимизатор включил в план запроса. Производительность возросла в 2 раза.