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

Отчёт по лабораторной работе №1

Дисциплина: Базы данных

Оптимизация SQL-запросов

Выполнил студент гр. 43501/4 _____Шаляпин Н.С

Преподаватель: _____Мяснов А.В.

Санкт-Петербург
2016

Цели 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

2. Способы оптимизации SQL-запросов

Индексы

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.

Создание индексов:

```
CREATE INDEX <index_name> ON <table_name> (<column_name> [DESC],…)
```

Модификация запросов

Явное указание порядка обхода таблиц с помощью конструкций `JOIN`.

Планы выполнения запросов

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 БД

Введение функциональной избыточности (дополнительных полей) для ускорения запросов.

3. Выполнение работы

1. **Проведено ознакомление со способами оптимизации SQL-запросов с использованием:**
    a. индексов
    b. модификации запроса
    c. создания собственного плана запроса
    d. денормализации БД
2. **Нагенерированы данные во всех таблицах.**

3. Оценивание производительности запроса.

Вид запроса: Вывести топ-10 моделей по продажам за заданный период.

```
select first 10 marks.mark as Marki, model_car.model_car as Model,
COUNT(trade.trade_id) as Buyed from model_car, marks, trade ,car
where model_car.model_car_id = car.model_car and car.car_id = trade.car_id and
marks.marks_id = car.marks_id and
trade.data_trade between '2016-01-01' and '2017-01-01'
group by model_car.model_car, marks.mark order by Buyed desc;
```

Результат запроса:

MARKI	MODEL	BUYED
WH)aG/bO57&KtH$	=hC	8
aFd8N#k	51	8
9	\[	8
6	E)t|U2l	7
a-HI-	W7▯`\d)6c	7
OV#▯o)(Yp#~a:	UXZ0@ 9~p}}GQ/n(	7
v0@_:!GA}BHR;NT\	2i	6
tm$hxX0]'▯	0!hl	6
'N7o35&uqzupFC3-	1JGt'PP3[]!)'Gh	6
f`,b%,`	-KW)FPE16	6

Скрипт выполняется на большом объеме данных(~140000 записей в таблице).
Время выполнения запроса.

№ Запроса	Время выполнения запроса
1	2s 153ms
2	2s 168ms
3	2s 152ms
4	2s 168ms
5	2s 156ms
Среднее	2s 159,4ms

4. Выполнение оптимизации

Изменим запрос с использованием конструкции join.

```
select first 10 tb3.mark as Marki, model_car.model_car as Model,
COUNT(tb2.trade_id) as Buyed from model_car

join car tb1 on model_car.model_car_id = tb1.model_car
join trade tb2 on tb1.car_id = tb2.car_id
join marks tb3 on tb3.marks_id = tb1.marks_id

where
--model_car.model_car_id = car.model_car
--and marks.marks_id = car.marks_id
--and car.car_id = trade.car_id and
tb2.data_trade between '2016-01-01' and '2017-01-01'
group by model_car.model_car, tb3.mark order by Buyed desc;
```

Результат запроса:

MARKI	MODEL	BUYED
WH)aG/bO57&KtH$	=hC	8
aFd8N#k	51	8
9	\[	8
6	E)t|U2l	7
a-HI-	W7▯`\d)6c	7
OV#▯b)(Yp#~a:	UXZ0@ 9~p}}GQ/n(	7
v0@_:!GA}BHR;NT\	2i	6
tm$hxX0]'▯	0!hl	6
"N7o35&uqzupFC3-	1JGt"PP3l]!)'Gh	6
f`,b%,`	-KW)FPE16	6

Время выполнения запроса.

№ Запроса	Время выполнения запроса
1	2s 28ms
2	2s 43ms
3	2s 90ms
4	2s 59ms
5	2s 106ms
Среднее	2s 65.2ms

Время выполнения уменьшилось на 94.2ms.

Добавим индексы в таблицы eat, deliveries, linkprod, kind для атрибутов amount, cost, amount, name:

```
create index  index_marks  on marks(mark);
create index  index_trade  on trade(data_trade);
create index  index_model_car  on model_car(model_car);
```

Запрос:

```
select first 10 marks.mark as Marki, model_car.model_car as Model,
COUNT(trade.trade_id) as Buyed from model_car, marks, trade ,car
where model_car.model_car_id = car.model_car and car.car_id = trade.car_id and
marks.marks_id = car.marks_id and
trade.data_trade between '2016-01-01' and '2017-01-01'
group by model_car.model_car, marks.mark order by Buyed desc;
```

Результат запроса:

MARKI	MODEL	BUYED
WH)aG/bO57&KtH$	=hC	8
aFd8N#k	51	8
9	\[	8
6	E)t|U2l	7
a-HI-	W7▯`\d)6c	7
OV#▯b)(Yp#~a:	UXZ0@ 9~p}}GQ/n(	7
v0@_:!GA}BHR;NT\	2i	6
tm$hxX0]'▯	0!hl	6
'N7o35&uqzupFC3-	1JGt'PP3l]!)'Gh	6
f`,b%,`	-KW)FPE16	6

Время выполнения запроса.

№ Запроса	Время выполнения запроса
1	1s 934ms
2	1s 919ms
3	1s 950ms
4	1s 950ms
5	1s 934ms
Среднее	1s 937.6ms

Время выполнения запроса уменьшилось на 221.8 ms.

Теперь изменим запрос с использованием конструкции join(индексы присутствуют).

```
select first 10 tb3.mark as Marki, model_car.model_car as Model,
COUNT(tb2.trade_id) as Buyed from model_car

join car tb1 on model_car.model_car_id = tb1.model_car
join trade tb2 on tb1.car_id = tb2.car_id
join marks tb3 on tb3.marks_id = tb1.marks_id

where
--model_car.model_car_id = car.model_car
--and marks.marks_id = car.marks_id
--and car.car_id = trade.car_id and
tb2.data_trade between '2016-01-01' and '2017-01-01'
group by model_car.model_car, tb3.mark order by Buyed desc;
```

Время выполнения запроса.

№ Запроса	Время выполнения запроса
1	1s 982ms
2	1s 981ms
3	1s 996ms
4	1s 965ms
5	1s 981ms
Среднее	1s 981ms

Время выполнения запроса уменьшилось на 178.4ms, но этот результат хуже, чем в предыдущем опыте с использованием только индексов. Зато производительность запроса отдельно с конструкцией join заметно ниже.

Нарушим первое правило нормализации: отомарность.

Создадим в таблице trade ещё один столбец data_trade_where_const , в котором будут отмечены 1 только те поля, которые удовлетворяют условию where trade.data_trade between '2016-01-01' and '2017-01-01'.

```
alter table trade
add   data_trade_where_const date;
update trade
set  data_trade_where_const=1
where trade.data_trade between '2016-01-01' and '2017-01-01';
```

Запрос в денормализованную таблицу

```
select first 10 marks.mark as Marki, model_car.model_car as Model,
COUNT(trade.trade_id) as Buyed from model_car, marks, trade ,car
where model_car.model_car_id = car.model_car and car.car_id = trade.car_id and
marks.marks_id = car.marks_id and
trade.data_trade_where_const = 1
group by model_car.model_car, marks.mark order by Buyed desc;
```

Результат запроса:

MARKI	MODEL	BUYED
WH)aG/bO57&KtH$	=hC	8
aFd8N#k	51	8
9	\[	8
6	E)t|U2l	7
a-HI-	W7▯`\d)6c	7
OV#▯o)(Yp#~a:	UXZ0@ 9~p}}GQ/n(	7
v0@_:!GA}BHR;NT\	2i	6
tm$hxX0]'▯	0!hl	6
'N7o35&uqzupFC3-	1JGt'PP3[]!)'Gh	6
f`,b%,`	-KW)FPE16	6

Время выполнения запроса.

№ Запроса	Время выполнения запроса
1	1s 872ms
2	1s 887ms
3	1s 872ms
4	1s 872ms
5	1s 888ms
Среднее	1s 878,2ms

Время выполнения запроса улучшилось, теперь запрос выполняется на 102.2ms быстрее.

Выводы

Произведена попытка оптимизировать запрос тремя способами: индексами, денормализацией и изменением структуры запроса. К уменьшению времени запроса привели все три способа. При совместном использовании модификации запроса и индексов результат оказался хуже, чем только с индексами.

Использование денормализации помогло ускорить выполнение запроса на 102.2ms, что в данных условиях стало наилучшим вариантом для оптимизации. Хотя в некоторых случаях использование денормализации может быть необходимо для повышения производительности, нужно помнить, что денормализация увеличивает число повторений данных, следовательно, возрастает используемое дисковое пространство, а также усложняется изменение БД при изменении окружающей среды.

Использование индексов также позволило снизить время выполнения запроса. Индексы ускоряют поиск данных, но замедляют операции добавления, удаления и модификации данных в индексируемых столбцах, поскольку при каждом изменении данных в такой таблице приходится перестраивать индекс. При создании большого количества индексов, размер индексного файла может быстро достичь максимального размера.

Таким образом, мы научились простым способам оптимизации запросов.