Санкт-Петербургский государственный политехнический университет

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе №1**

**Дисциплина**: Базы данных

**Оптимизация SQL-запросов**

Выполнил студент гр. 43501/3 Муравьев Ф.Э.

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

### Индексы

Индекс – объект БД создаваемый специально для повышения производительности запросов. Индекс обеспечивает эффективное упорядочивание записей таблицы по заданному полю (или полям) для указанного порядка. В индексе хранится местоположение записей на основе одного или нескольких полей, которые являются частью индекса. Индекс может быть гораздо меньше описываемой им таблицы (это зависит от количества уникальных значений проиндексированных полей), делая чтение данных более эффективным.

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.

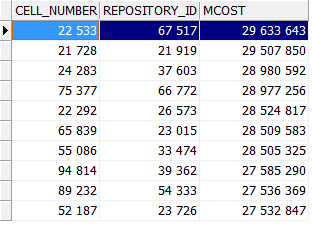
**3. Выполнение работы**

1. **Провелено ознакомление со способами оптимизации SQL-запросов с использованием:** 
   1. индексов
   2. модификации запроса
   3. создания собственного плана запроса
   4. денормализации БД
2. **Нагенерированы данные во всех таблицах.**
3. **Оценивание производительности запроса.**

Вид запроса: **10 ячеек с max суммарной стоимостью.**

|  |
| --- |
| select first 10 cells.cell\_number, repository\_id, sum(items.cost) as MCost from cells, items, history\_cell  where (history\_cell.cell\_number=cells.cell\_number)and(history\_cell.item\_id=items.item\_id)and(history\_cell.acsess\_type=1)and(history\_cell.item\_id not in (select item\_id from history\_cell where item\_id=2)) group by cells.cell\_number, cells.repository\_id order by MCost desc; |

Результат запроса:



При выполнении на большом объеме данных(100000 записей в таблице) время выполнения скрипта – 921ms.

Время выполнения запроса.

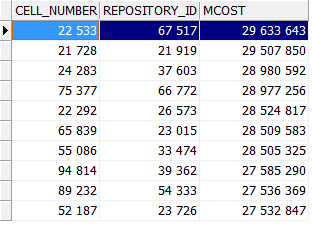
|  |  |
| --- | --- |
| № Запроса | Время выполнения запроса |
| 1 | 921ms |
| 2 | 938ms |
| 3 | 953ms |
| 4 | 969ms |
| 5 | 938ms |
| Среднее | 943.8ms |

1. **Выполнение оптимизации**

Изменим запрос с использованием конструкции join.

|  |
| --- |
| select first 10 cells.cell\_number, repository\_id, sum(items.cost) as MCost from history\_cell  join items on history\_cell.item\_id=items.item\_id  join cells on history\_cell.cell\_number=cells.cell\_number  where (history\_cell.acsess\_type=1)and(history\_cell.item\_id not in (select item\_id from history\_cell where item\_id=2)) group by cells.cell\_number, cells.repository\_id order by MCost desc; |

Результат запроса:



При выполнении на большом объеме данных(100000 записей в таблице) время выполнения скрипта – 937ms.

Время выполнения запроса.

|  |  |
| --- | --- |
| № Запроса | Время выполнения запроса |
| 1 | 937ms |
| 2 | 968ms |
| 3 | 969ms |
| 4 | 985ms |
| 5 | 969ms |
| Среднее | 965.6 ms |

Время выполнения немного увеличилось.

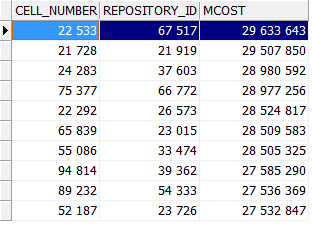
**Добавим индексы в таблицы cells, history\_cell для атрибутов item\_id, cell\_number, acsess\_type:**

|  |
| --- |
| create index ind\_itemid on history\_cell (item\_id);  create index ind\_acsesstype on history\_cell (acsess\_type);  create index ind\_cellnumber on history\_cell (cell\_number);  create index ind\_cellnumber\_cells on cells (cell\_number); |

Запрос:

|  |
| --- |
| select first 10 cells.cell\_number, repository\_id, sum(items.cost) as MCost from cells, items, history\_cell  where (history\_cell.cell\_number=cells.cell\_number)and(history\_cell.item\_id=items.item\_id)and(history\_cell.acsess\_type=1)and(history\_cell.item\_id not in (select item\_id from history\_cell where item\_id=2)) group by cells.cell\_number, cells.repository\_id order by MCost desc; |

Результат запроса:



При выполнении на большом объеме данных(100000 записей в таблице) время выполнения скрипта – 906ms.

Время выполнения запроса.

|  |  |
| --- | --- |
| № Запроса | Время выполнения запроса |
| 1 | 906ms |
| 2 | 907ms |
| 3 | 906ms |
| 4 | 906ms |
| 5 | 922ms |
| Среднее | 909.4 |

Время выполнения запроса уменьшилась на 34.4 ms.

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

|  |
| --- |
| select first 10 cells.cell\_number, repository\_id, sum(items.cost) as MCost from history\_cell  join items on history\_cell.item\_id=items.item\_id  join cells on history\_cell.cell\_number=cells.cell\_number  where (history\_cell.acsess\_type=1)and(history\_cell.item\_id not in (select item\_id from history\_cell where item\_id=2)) group by cells.cell\_number, cells.repository\_id order by MCost desc; |

При выполнении на большом объеме данных(100000 записей в таблице) время выполнения скрипта – 901.2ms.

Время выполнения запроса.

|  |  |
| --- | --- |
| № Запроса | Время выполнения запроса |
| 1 | 897ms |
| 2 | 907ms |
| 3 | 906ms |
| 4 | 890ms |
| 5 | 906ms |
| Среднее | 901.2ms |

Время выполнения запроса уменьшилась на 42.6ms.

**Создадим новую, денормализованную таблицу cells,** которая будет копией таблицы cells и в которой заранее вычислим суммарную стоимость ценностей в данной ячейке.

|  |
| --- |
| alter table cells  add cost\_items int;  update cells set cost\_items=RAND()\*30000000; |

Запрос в денормализованную таблицу

|  |
| --- |
| select first 10 cells.cell\_number, repository\_id, cost\_items as MCost from cells  order by MCost desc; |

Время выполнения запроса.

|  |  |
| --- | --- |
| № Запроса | Время выполнения запроса |
| 1 | 78ms |
| 2 | 78ms |
| 3 | 78ms |
| 4 | 78ms |
| 5 | 63ms |
| Среднее | 75.2 |

Время выполнения запроса улучшилось, теперь запрос выполняется на 868.6 ms быстрее.

Выводы.

Произведена попытка оптимизировать запрос тремя способами: индексами, денормализацией и изменением структуры запроса. В данном случае, к уменьшению времени запроса привело только использование индексов.

Изменение структуры запроса уменьшило время выполнения только в совокупности с введением индексов. Без индексов время выполнения запроса не менялось.

Ускорение с помощью денормализованной таблицы прошло успешно. Запрос стал выполнятся быстрее на 868.6 ms.

Таким образом, мы научились простым способам оптимизации запросов.