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

ОТЧЕТ №1
по дисциплине «Базы данных»
Оптимизация запросов

Студент гр.43501/4
Алексеев Д.М.

Преподаватель
Мяснов А.В.

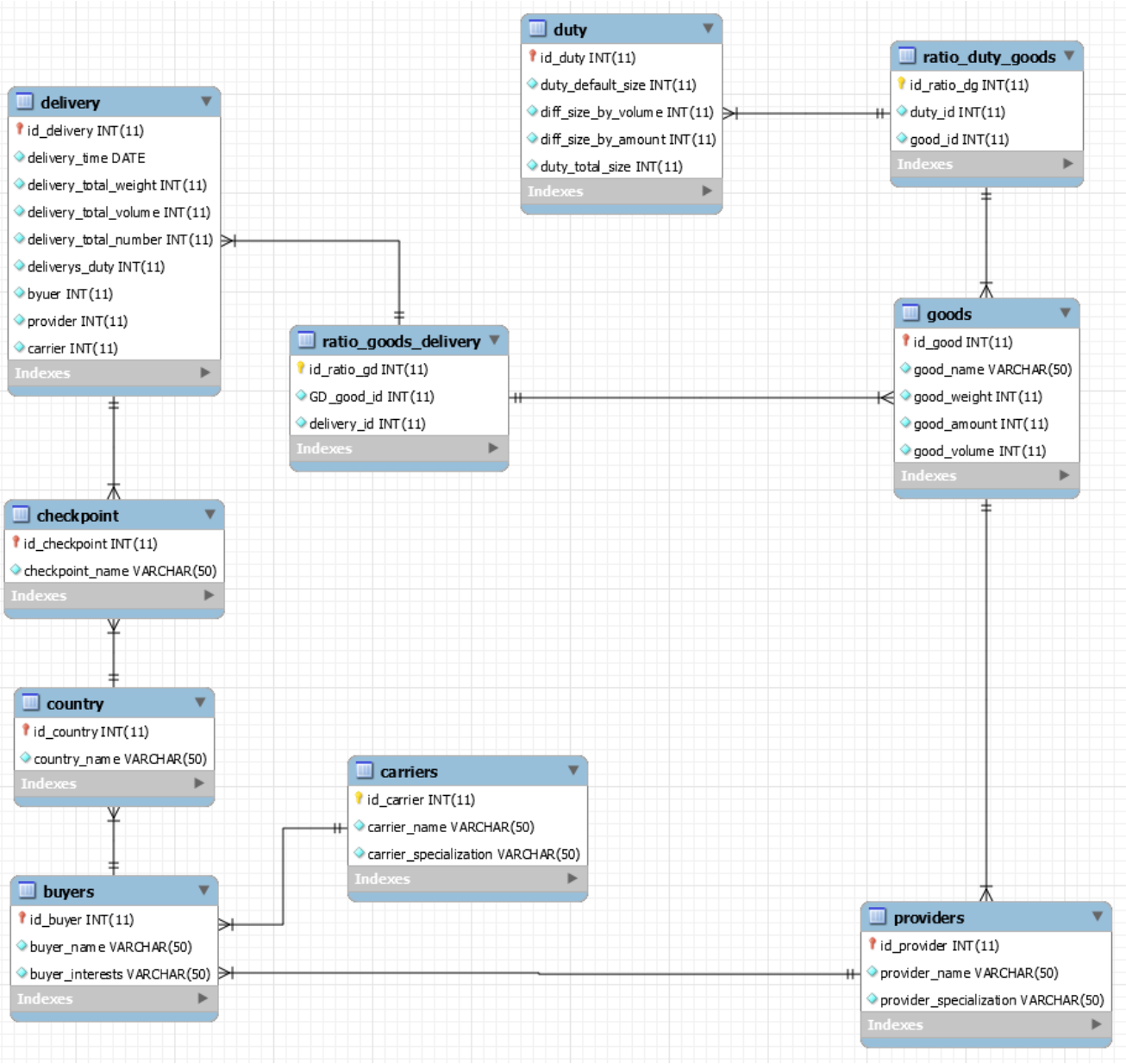
Санкт-Петербург
2016 год

1. Цель работы
   Получить практические навыки создания эффективных SQL-запросов.

2. 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
   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

   Выполнить следующий запрос:

   Вывести 10 типов товаров, за которые были получены максимальные суммарные пошлины за заданный промежуток времени с суммарной величиной пошлин.



3. Выполнение задания

Вывести 10 типов товаров, за которые были получены максимальные суммарные пошлины за заданный промежуток времени с суммарной величиной пошлин.

Простой запрос:

Group_TEN_by_max_duty.sql

```
SELECT goods.good_name, SUM(duty.duty_total_size) FROM duty, ratio_duty_goods,
goods,  delivery, ratio_goods_delivery
#Сначала связываем товар с пошлиной,
WHERE duty.id_duty = ratio_duty_goods.duty_id AND ratio_duty_goods.good_id =
goods.id_good
#потом - товар с поставкой.
AND ratio_goods_delivery.GD_good_id = goods.id_good AND
ratio_goods_delivery.delivery_id = delivery.id_delivery
#Задаём время.
AND (DATE(delivery_time) BETWEEN '2010-01-01' AND '2015-12-30')
GROUP BY goods.good_name #Группируем по имени
HAVING SUM(duty.duty_total_size)>0
ORDER BY SUM(duty.duty_total_size) DESC
```

```
LIMIT 10; #Выводим всего 10
```

Включим профилирование (по-умолчанию оно выключено), кол-во показываемых сообщений - 2:

```
SET profiling=1;
SET profiling_history_size = 2;
```

Теперь сделаем наш запрос; он идёт очень долго – прервём его выполнение и выведем список запросов:

```
SHOW profiles;
```

Результат:

Результат #1 (3×2)

Query_ID	Duration	Query
10	0,00006300	SHOW WARNINGS LIMIT 5
11	5,05960650	SELECT goods.good_name, SUM(duty.duty_total_size) F...

WARNING – это было сообщение, что profiling является устаревшим (deprecated) и скоро будет выпилен. Нас интересует вторая строка: это наш запрос.
Поля у таблицы следующие:

- Query_ID – номер запроса в текущей сессии.
- Duration – длительность выполнения операции
- Query – это наш запрос (его текстовое описание)

Хоть мы и не дождались выполнения запроса, мы получили время его обработки – чуть больше 5 секунд.

Посмотрим, как будет производиться операция SELECT (порядок и метод связывания таблиц): просто перед запросом поставим оператор EXPLAIN:

```
EXPLAIN SELECT goods.good_name, SUM(duty.duty_total_size) FROM duty,
ratio_duty_goods, goods,  delivery, ratio_goods_delivery
…
```

Результат:

Результат #1 (10×5)

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	ratio_duty_goods	ALL	(NULL)	(NULL)	(NULL)	(NULL)	100 098	Using temporary; Using filesort
1	SIMPLE	duty	eq_ref	PRIMARY	PRIMARY	4	customs_zero.ratio_duty_goods.duty_id	1	(NULL)
1	SIMPLE	goods	eq_ref	PRIMARY	PRIMARY	4	customs_zero.ratio_duty_goods.good_id	1	(NULL)
1	SIMPLE	ratio_goods_delivery	ALL	(NULL)	(NULL)	(NULL)	(NULL)	100 098	Using where; Using join buffer (Block Nested Loop)
1	SIMPLE	delivery	eq_ref	PRIMARY	PRIMARY	4	customs_zero.ratio_goods_delivery.delivery_id	1	Using where

В полученной таблице есть следующие поля[1][2]:

- **table** Таблица, к которой относится выводимая строка.
- **type** Тип связывания.
  - ✓ **ALL** Для каждой комбинации строк из предыдущих таблиц будет производиться полный просмотр этой таблицы. Это обычно плохо, если таблица - первая из не отмеченных как const, и очень плохо во всех остальных случаях. Как правило, можно избегать типа связывания ALL - путем добавления большего количества индексов таким образом, чтобы строка могла быть найдена при помощи константных значений или значений столбца из предыдущих таблиц.
  - ✓ **eq_ref** Для каждой комбинации строк из предыдущих таблиц будет считываться одна

строка из этой таблицы. Это наилучший возможный тип связывания среди типов, отличных от const. Данный тип применяется, когда все части индекса используются для связывания, а сам индекс - UNIQUE или PRIMARY KEY.

- **possible_keys** Столбец possible_keys служит для указания индексов, которые может использовать MySQL для нахождения строк в этой таблице. Обратите внимание: этот столбец полностью независим от порядка таблиц. Это означает, что на практике некоторые ключи в столбце possible_keys могут не годиться для сгенерированного порядка таблиц. Если данный столбец пуст, то никаких подходящих индексов не имеется. В этом случае для увеличения производительности следует исследовать выражение WHERE, чтобы увидеть, есть ли в нем ссылки на какой-либо столбец (столбцы), которые подходили бы для индексации. Если да, создайте соответствующий индекс и снова проверьте запрос при помощи оператора EXPLAIN.
- **key** Столбец key содержит ключ (индекс), который MySQL решил использовать в действительности. Если никакой индекс не был выбран, ключ будет иметь значение NULL.
- **key_len** Столбец key_len содержит длину ключа, которую решил использовать MySQL. Если key имеет значение NULL, то длина ключа (key_len) тоже NULL. Обратите внимание: по значению длины ключа можно определить, сколько частей составного ключа в действительности будет использовать MySQL.
- **ref** Столбец ref показывает, какие столбцы или константы используются с ключом, указанным в key, для выборки строк из таблицы.
- **rows** В столбце rows указывается число строк, которые MySQL считает нужным проанализировать для выполнения запроса.
- **Extra** Этот столбец содержит дополнительную информацию о том, как MySQL будет выполнять запрос.
  - ✓ **Using temporary** Чтобы выполнить запрос, MySQL должен будет создать временную таблицу для хранения результата. Это обычно происходит, если предложение ORDER BY выполняется для набора столбцов, отличного от того, который используется в предложении GROUP BY.
  - ✓ **Using filesort** MySQL должен будет сделать дополнительный проход, чтобы выяснить, как извлечь строки в порядке сортировки. Для выполнения сортировки выполняется просмотр всех строк согласно типу связывания (join type) и сохраняются ключ сортировки + указатель на строку для всех строк, удовлетворяющих выражению WHERE. После этого ключи сортируются и строки извлекаются в порядке сортировки.
  - ✓ **Using where** Ограничиваем строки, которые соответствуют следующей таблице.
  - ✓ **Using join buffer** Таблицы из предыдущих команд Join складываются в буффер, и их значения читаются с буфера, чтобы выполнять последующие команды Join с ними.
  - ✓ **NULL** Метод поиска по индексу невозможен

Теперь, согласно заданию, попробуем оптимизировать запрос несколькими способами.

Модификация запроса, используя JOIN.
Group_TEN_by_max_duty_with_JOIN.sql

```
SELECT goods.good_name, SUM(duty.duty_total_size)
FROM goods
#Сначала связываем товар с пошлиной,
INNER JOIN  ratio_duty_goods
ON goods.id_good = ratio_duty_goods.good_id
LEFT JOIN duty
ON ratio_duty_goods.duty_id = duty.id_duty
#потом связываем товар с поставкой.
```

```
INNER JOIN  ratio_goods_delivery
ON goods.id_good = ratio_goods_delivery.GD_good_id
LEFT JOIN delivery
ON ratio_goods_delivery.delivery_id = delivery.id_delivery
#Уточняем время
WHERE (DATE(delivery_time) BETWEEN '2010-01-01' AND '2015-12-30')
GROUP BY goods.good_name #Группируем по имени
HAVING SUM(duty.duty_total_size)>0
ORDER BY SUM(duty.duty_total_size) DESC
LIMIT 10; #Выводим всего 10
```

Хоть мы и не дождались выполнения запроса, мы получили время его обработки – чуть больше 11 секунд:

Результат #1 (3×2)

Query_ID	Duration	Query
7	0,00007825	SHOW WARNINGS LIMIT 5
8	11,73653350	SELECT goods.good_name, SUM(duty.duty_total_size)...

Используем индексы.

Для повышения быстродействия создадим индексы (made_indices.sql):

```
CREATE INDEX delivery_time ON delivery(delivery_time);
CREATE INDEX duty_id ON ratio_duty_goods(duty_id);
CREATE INDEX good_id ON ratio_duty_goods(good_id);
CREATE INDEX GD_good_id ON ratio_goods_delivery(GD_good_id);
CREATE INDEX delivery_id ON ratio_goods_delivery(delivery_id);
```

Обычный SQL-запрос (без JOIN):

```
SELECT goods.good_name, SUM(duty.duty_total_size) FROM duty, ratio_duty_goods,
goods,  delivery, ratio_goods_delivery
#Сначала связываем товар с пошлиной,
WHERE duty.id_duty = ratio_duty_goods.duty_id AND ratio_duty_goods.good_id =
goods.id_good
#потом - товар с поставкой.
AND ratio_goods_delivery.GD_good_id = goods.id_good AND
ratio_goods_delivery.delivery_id = delivery.id_delivery
#Задаём время.
AND (DATE(delivery_time) BETWEEN '2010-01-01' AND '2015-12-30')
GROUP BY goods.good_name #Группируем по имени
HAVING SUM(duty.duty_total_size)>0
ORDER BY SUM(duty.duty_total_size) DESC
LIMIT 10; #Выводим всего 10
```

Результат:

Query_ID	Duration	Query
35	1,39714400	SELECT goods.good_name, SUM(duty.duty_total_size) F...

Чуть больше, чем за секунду, мы получили результат на тот запрос, который не могли выполнить ранее:

goods (2×10)

good_name	SUM(duty.duty_total_size)
Good_591	1 852
Good_4795	1 843
Good_4479	1 842
Good_4924	1 836
Good_3728	1 832
Good_4164	1 827
Good_2728	1 824
Good_191	1 820
Good_3225	1 820
Good_1739	1 819

4. Выводы

На этой работе мы познакомились со способами оценки быстродействия запросов, их анализа и повышения быстродействия.

Один из способов – добавление индексов. Он заметно повышает быстродействие запроса, но имеет следующие недостатки: индексы занимают место в памяти и модификация данных в столбцах с индексами занимает больше времени.

Другой способ – модификация запроса. К сожалению, модифицировать запрос таким образом, чтобы получить повышение быстродействия, не получилось.

Не использованы были ещё 2 способа: денормализация (введение функциональной избыточности для ускорения запроса) и создание собственного плана выполнения запроса (на основе таблицы, полученной из EXPLAIN; *данный способ не рассматривался. т.к. в MySQL он сводится к правильному созданию индексов*).

Использованные источники

[1] http://www.mysql.ru/docs/man/EXPLAIN.html

[2] http://dev.mysql.com/doc/refman/5.6/en/explain-output.html#explain-extra-information