Курс: Основы реляционных баз данных. MySQL
Урок 11. Оптимизация запросов. NoSQL
Выполнил: Кузнецов Сергей (Факультет Geek University Python-разработки)

Домашнее задание:

Практическое задание по теме “Оптимизация запросов”

1. Создайте таблицу logs типа Archive. Пусть при каждом создании записи в таблицах users, catalogs и products в таблицу logs помещается время и дата создания записи, название таблицы, идентификатор первичного ключа и содержимое поля name.
2. (по желанию) Создайте SQL-запрос, который помещает в таблицу users миллион записей.

Практическое задание по теме “NoSQL”

(опционально, сделаю позже, т.к. сейчас 2 курса параллельно и на работе «завал»)

3. В базе данных Redis подберите коллекцию для подсчета посещений с определенных IP-адресов.
4. При помощи базы данных Redis решите задачу поиска имени пользователя по электронному адресу и наоборот, поиск электронного адреса пользователя по его имени.
5. Организуйте хранение категорий и товарных позиций учебной базы данных shop в СУБД MongoDB.

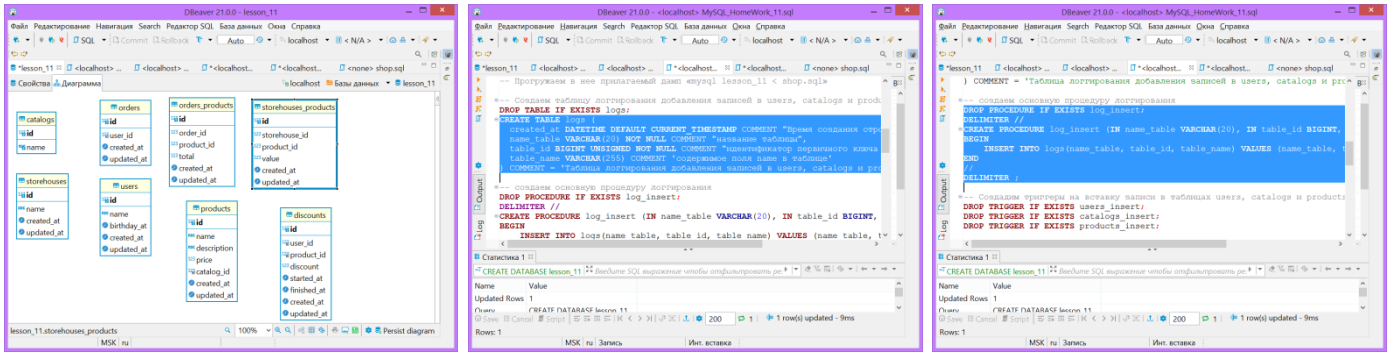
---

1. Создайте таблицу logs типа Archive. Пусть при каждом создании записи в таблицах users, catalogs и products в таблицу logs помещается время и дата создания записи, название таблицы, идентификатор первичного ключа и содержимое поля name.

Создаем БД «*CREATE DATABASE lesson_11;*». В принципе, можно все и в БД «lesson_08», но потренируемся в копировании БД. Прогружаем в нее прилагаемый дамп «*mysql lesson_11 < shop.sql*».

Создаем таблицу логгирования добавления записей в users, catalogs и products

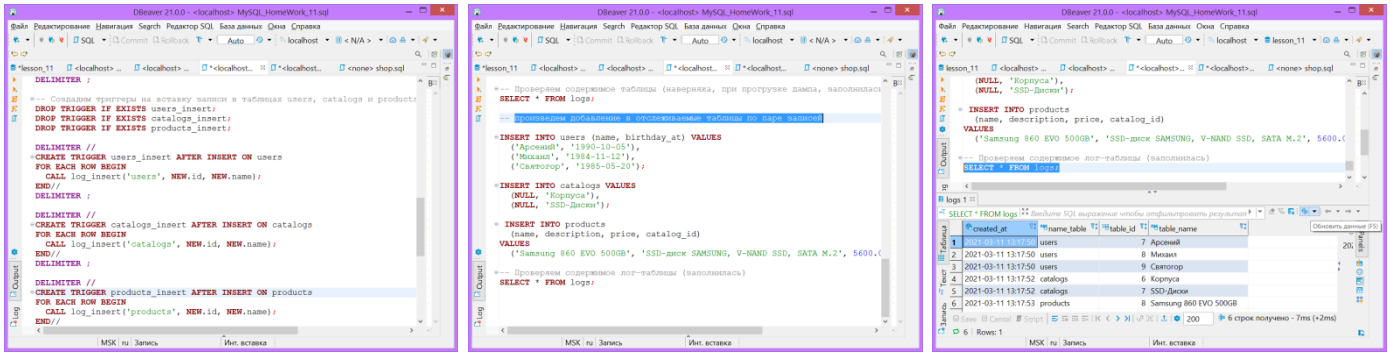
Создаем основную процедуру логгирования



Активируем три триггера на вставку строк в таблицы users, catalogs и products.

Произведем добавление в отслеживаемые таблицы по паре записей.

Проверяем содержимое лог-таблицы (заполнилась) – цель достигнута.



2. Создайте SQL-запрос, который помещает в таблицу users миллион записей.

Вариант 1 (все одним запросом, без вспомогательных таблиц)

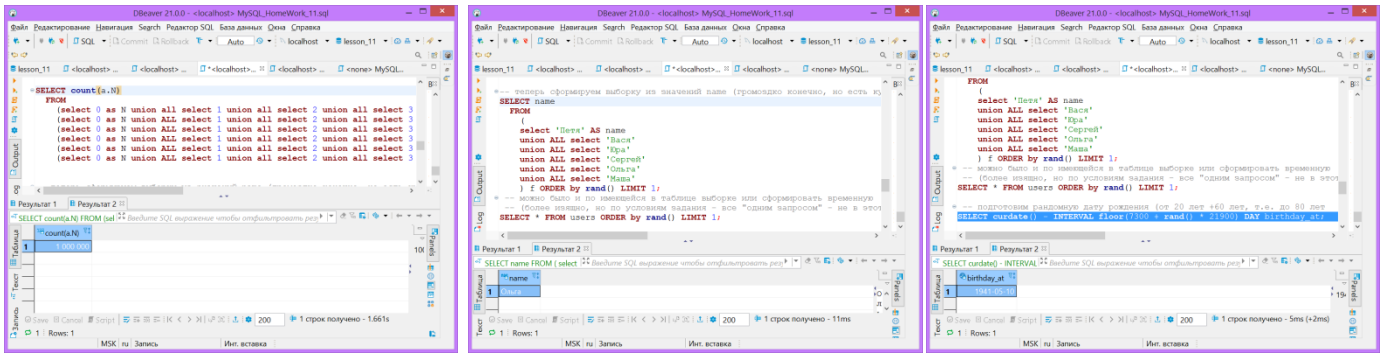
Сформируем SELECT дающий 1млн записей. (т.к. такой большой таблицы у нас нет "под рукой"). Воспользуемся объединением (union all), для получения 10 значений и пересечением для умножения на такой-же набор 10... Нам потребуется 10*10..*10(6 раз) для получения 1 000 000. Проверяем – все верно, возвращает 1млн записей.

После этого, сформируем выборку из значений name (громоздко конечно, но есть куда стремиться)

```
SELECT name
 FROM
        (
        select 'Петя' AS name
        union ALL select 'Вася'
        union ALL select 'Юра'
        union ALL select 'Сергей'
        union ALL select 'Ольга'
        union ALL select 'Маша'
        ) f ORDER by rand() LIMIT 1;
```

Можно было и по имеющейся в таблице выборке или сформировать временную таблицу и брать из нее *(более изящно, но по условиям задания - все "одним запросом" , так что не в этот раз)* «*SELECT * FROM users ORDER by rand() LIMIT 1;*»

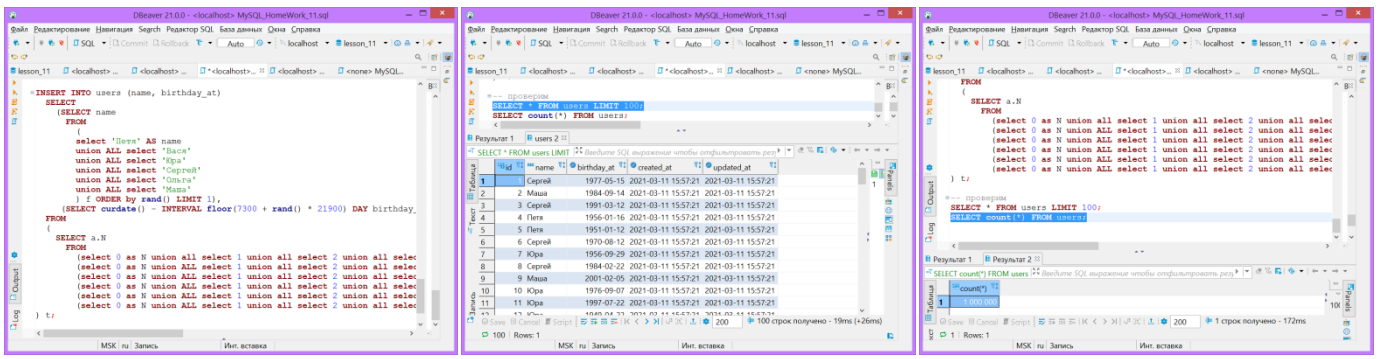
Подготовим рандомную дату рождения (от 20 лет +60 лет, т.е. до 80 лет) «*SELECT curdate() - INTERVAL floor(7300 + rand() * 21900) DAY birthday_at;*»



Пришло время сконструировать итоговый INSERT на «1 000 000» записей. Конечно, по-хорошему, нужно отключить индексы на период выполнения запроса, чтобы не потратить много времени… и... сначала тестируем на 10 записях – все ок. Для чистоты, обнулим таблицу и счетчики «*TRUNCATE users;*»

Запрос выполнялся (1000000 row(s) updated - 1m 48s). Проверим «*SELECT * FROM users LIMIT 100;*»

Уточним «*SELECT count(*) FROM users;*» - все верно 1000000 записей!



Посмотрим, что с логами - тоже добавлено 1 000 000 записей «*SELECT count(*) FROM logs;*»,

но все от одного времени создания, т.е. оптимизатор не растягивал "удовольствие" на 1млн попыток

«*SELECT * FROM LOGS LIMIT 990000,100;*»

Вариант 2 (с использованием вспомогательной таблицы)

Создадим таблицу для хранения вариантов имен (не менее 10 строк).
«*CREATE TABLE name_v (name VARCHAR(20) NOT NULL);*»
Временная таблица не подходит ввиду ограничения mySQL "*Can't reopen table*" при попытке использовать таблицу более одного раза в рамках одного запроса.
Добавим в нее несколько значений:
INSERT INTO name_v VALUES ('Василий'), ('Юрий'), ('Сергей'), ('Ольга'), ('Маша'), ('Пётр'), ('Борис'), ('Иннокентий'), ('Наталья'), ('Ксения'), ('Изабелла'), ('Иосиф');
По аналогии с первым вариантом, сформируем итоговый запрос *(выглядит короче и понятнее)*:

```
INSERT INTO users (name, birthday_at)
 SELECT
        (SELECT name FROM name_v ORDER by rand() LIMIT 1),
        (SELECT curdate() - INTERVAL floor(7300 + rand() * 21900) DAY birthday_at)
 FROM
 (
        SELECT a.N
          FROM
                (SELECT 1 AS N FROM name_v LIMIT 10) a,
                (select 2 FROM name_v LIMIT 10) b,
                (select 3 FROM name_v LIMIT 10) c,
                (select 4 FROM name_v LIMIT 10) d,
                (select 4 FROM name_v LIMIT 10) e,
                (select 5 FROM name_v LIMIT 10) f
) t;
```

Запрос выполнялся (1000000 row(s) updated - 2m 21s). Проверим «*SELECT * FROM users LIMIT 100;*»

Уточним «*SELECT count(*) FROM users;*» - все верно 1000000 записей!

Вариант 3 (с использованием процедуры)

Самый простой, с т.з. логики вариант. Создаем процедуру:

```
CREATE PROCEDURE proc_insert (IN num_str int)
BEGIN
 DECLARE v1 INT DEFAULT 1;
 WHILE v1 <= num_str DO
        INSERT INTO users (name, birthday_at) VALUES (
                (SELECT name FROM name_v ORDER by rand() LIMIT 1),
                (SELECT curdate() - INTERVAL floor(7300 + rand() * 21900) DAY birthday_at));
  SET v1 = v1 + 1;
 END WHILE;
END
```

Запускаем на исполнение «*CALL proc_insert(1000000);*»

За час работы процедура вставила порядка 500тыс записей. При этом, система практически зависла (ноут слабенький) и пришлось произвести перезагрузку, т.к. сервер не освободил фоновые процессы.

По итогу «жесткой» перезагрузки, выяснилось, что DBeaver похерил собственную базу (при входе отказывался работать и отправлял в ЛОГ-файл). Также, он уничтожил скрипт с домашней работой (обнулил)… Пришлось переустановить DBeaver и «вручную» переписать скрипт (часть, конечно, восстановил из описаний, сохранившихся в БД – DDL редактор).

Выводы:

- Быстрее всего отработал запрос с минимум чтений из БД, работающий только на добавление записей (1m 48s)
- Чуть медленнее, отработал запрос с использованием вспомогательной таблицы (2m 21s). Он менее громоздкий и более удобный.
- Вариант с процедурой и построчным добавлением, хоть и самый элегантный, не очень жизненный и для его выполнения нужны существенные ресурсы, как сервера, так и времени (порядка 2-х часов). Нужно было отключить логгирование в файл «logs». В него также было добавлено миллион записей!!!
- Когда-то придется разобраться в управлении выделением памяти MySQL и эффективностью кэша InnoDB