МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Лабораторна робота №13
з курсу
«ОБДЗ»
на тему:
«Написання збережених процедур на мові SQL»

Виконав:
Студент групи КН-209
Гладун Ярослав

Львів-2020

Лабораторна робота №13

Тема: Аналіз та оптимізація запитів.

Мета: Навчитися аналізувати роботу СУБД та оптимізувати виконання складних запитів на вибірку даних. Виконати аналіз складних запитів за допомогою директиви EXPLAIN, модифікувати найповільніші запити з метою їх пришвидшення.

Хід роботи:
1. Для цієї лабораторної роботи було створено спеціальну базу даних та таблицю:

```
CREATE TABLE user
(
    id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
    name VARCHAR(300) NOT NULL,
    email VARCHAR(45) NOT NULL,
    phone VARCHAR(45) NOT NULL
);
```

```
INSERT INTO user (name, email, phone) VALUES ('Yaroslav', 'gladi41414@gmail.com', '380444345256');
```

2. Далі, щоб протестувати роботу індексів, потрібно було заповнити таблицю відносно великою кількістю даних. Вдалося це зробити наступним чином:

```
INSERT INTO user (name, email, phone)
SELECT CONCAT(name, LPAD(CONV(FLOOR(RAND()*POW(36, 6)), 10, 36), 6, 0)) as name,
       email,
       CONCAT(LEFT(phone, 8), ROUND(RAND()*100)) AS phone
FROM user;
```

За допомогою цього коду всі стрічки в таблиці постійно видозмінювалася та додавалися до неї ж. Кінцева кількість рядків вийшла 524288, а отже даний код був виконаний log2(524288) = 19 разів.

3. Далі було створено індекс на колонки name та phone. Отож поглянемо на вираш в часі при селекті.

```
SELECT * FROM user WHERE name='YaroslavFIGHL676QE13Y6CLU1S88B5G7RI5Y206SVYK0VT6AET2VA8W0WACS3';
```

- результат без індексу на name

```
1 row retrieved starting from 1 in 222 ms (execution: 207 ms, fetching: 15 ms)
```

- результата з індексом на name

```
1 row retrieved starting from 1 in 73 ms (execution: 4 ms, fetching: 69 ms)
```

Як бачимо, виграш в часі досить великий.

4. Щоб перевірити який виграш в часі буде при join та group by створемо ще одну таблицю, яка буде до кожного номеру телефону ставити адрес.

```
CREATE TABLE addresses (
    id INT PRIMARY KEY AUTO_INCREMENT,
    phone VARCHAR(45) NOT NULL,
    address VARCHAR(45) NOT NULL
);
```

```
INSERT INTO addresses (phone, address)
WITH phones AS (SELECT DISTINCT phone FROM user)
SELECT phones.phone, CONV(FLOOR(RAND() * 99999999999999), 10, 36)
FROM phones;
```

5. Отже, виконаємо наступну команду з EXPLAIN щоб побачити як працює даний запит

```
EXPLAIN SELECT * FROM user
INNER JOIN addresses on addresses.phone = user.phone
WHERE name='YaroslavFIGHL676QE13Y6CLU1S88B5G7RI5Y206SVYK0VT6AET2VA8W0WACS3';
```

	id	select_type	table	partitions	type
1	1	SIMPLE	addresses	<null>	ALL
2	1	SIMPLE	user	<null>	ALL

```
1 row retrieved starting from 1 in 233 ms (execution: 195 ms, fetching: 38 ms)
```

Як бачимо, type у обох таблицях ALL, а отже join проходиться повністю по обох таблицях.

	id	select_type	table	partitions	type
1	1	SIMPLE	user	<null>	ref
2	1	SIMPLE	addresses	<null>	ref

```
1 row retrieved starting from 1 in 36 ms (execution: 2 ms, fetching: 34 ms)
```

З використанням індексів швидкість запиту досить сильно придшвидшилась.

Висновок: на даній лабораторній роботі я навчився аналізувати і оптимізувати виконання запитів. Для аналізу запитів було використано директиву EXPLAIN, а для оптимізації – модифікація порядку з’єднання таблиць і створення додаткових індексів.