ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ _____

ПРЕПОДАВАТЕЛЬ

Доцент				Бржезовский А. В.
должность, уч. степень, звание		подпись, дата		инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1

ИНДЕКСАЦИЯ ДАННЫХ

Вариант 5

по курсу: МЕТОДЫ И СРЕДСТВА ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ И ТЕХНОЛОГИЙ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	4128				Воробьев В.А.
			подпись, дата		инициалы, фамилия

Санкт-Петербург 2024

СОДЕРЖАНИЕ

1 Постановка задачи

Цель работы: изучить принципы использования индексации данных.

1.1 Задание

- Произвести генерацию и вставку тестовых данных в БД, выполнить запросы из ЛР 3..5 или аналогичные им, зафиксировать планы и время выполнения запросов, создать систему индексов для ускорения выполнения запросов, повторно выполнить запросы, зафиксировать планы и время выполнения.
- С помощью директивы hints в операторе select задать оптимизатору решения относительно использования индексов, алгоритмов соединения таблиц, сравнить планы выполнения с созданными оптимизатором. Варианты заданий приведены в ПРИЛОЖЕНИИ.

5 Вариант:

Создайте базу данных для хранения следующих сведений: ВУЗ, студент, группа, факультет, конференция, тема доклада, программа конференции.

2 Выполнение работы

Текст запросов представлен в Приложении, а также в репозитории GitHub (URI - https://github.com/vladcto/suai-labs/tree/1feac804866a924523979b3a271d293076b96bdf/6_semester/%D0%9C%D0%A1%D0%9F%D0%98%D0%A1%D0%A2/7).

Для работы была использована БД из предыдущих работ.

```
1  CREATE DATABASE IF NOT EXISTS conference_db_lab1;
2
3  USE conference_db_lab1;
4
5  CREATE TABLE IF NOT EXISTS university
6  (
7    id    INT PRIMARY KEY AUTO_INCREMENT,
8    name VARCHAR(100) NOT NULL UNIQUE
9  );
10
11 CREATE TABLE IF NOT EXISTS faculty
12 (
13   id             INT PRIMARY KEY AUTO_INCREMENT,
14   university_id  INT NOT NULL,
15   number         INT NOT NULL UNIQUE,
16   FOREIGN KEY (university_id) REFERENCES university (id)
17 );
18
19 CREATE TABLE IF NOT EXISTS uni_group
20 (
21   id          INT PRIMARY KEY AUTO_INCREMENT,
22   faculty_id  INT          NOT NULL,
23   name        VARCHAR(50) NOT NULL UNIQUE,
24   FOREIGN KEY (faculty_id) REFERENCES faculty (id)
25 );
26
27 CREATE TABLE IF NOT EXISTS student
28 (
29   id        INT PRIMARY KEY AUTO_INCREMENT,
30   group_id  INT NOT NULL,
31   name      VARCHAR(100),
32   CONSTRAINT fk_group_id FOREIGN KEY (group_id) REFERENCES
       uni_group (id)
33 );
```

```
34
35 CREATE TABLE IF NOT EXISTS conference
36 (
37   id      INT PRIMARY KEY AUTO_INCREMENT,
38   name    VARCHAR(100) NOT NULL DEFAULT 'Научная конференция',
39   place   VARCHAR(100) NOT NULL DEFAULT 'ГУАП',
40   theme   VARCHAR(255) NOT NULL
41 );
42
43 CREATE TABLE IF NOT EXISTS conference_session
44 (
45   id              INT PRIMARY KEY AUTO_INCREMENT,
46   conference_id   INT  NOT NULL,
47   start_time      TIME NOT NULL,
48   end_time        TIME NOT NULL,
49   date            DATE NOT NULL,
50   CONSTRAINT fk_conf_id_session FOREIGN KEY (conference_id)
       REFERENCES conference (id),
51   CONSTRAINT check_time CHECK (start_time < end_time)
52 );
53
54 CREATE TABLE IF NOT EXISTS topic
55 (
56   id          INT PRIMARY KEY AUTO_INCREMENT,
57   title       VARCHAR(255) NOT NULL,
58   session_id  INT          NOT NULL,
59   CONSTRAINT fk_session_id_topic FOREIGN KEY (session_id)
       REFERENCES conference_session (id)
60 );
61
62 CREATE TABLE IF NOT EXISTS authorship
63 (
64   author_id  INT NOT NULL,
65   topic_id   INT NOT NULL,
66   CONSTRAINT pk_authorship PRIMARY KEY (author_id, topic_id),
67   CONSTRAINT fk_author_id FOREIGN KEY (author_id) REFERENCES
       student (id) ON DELETE CASCADE,
68   CONSTRAINT fk_topic_id FOREIGN KEY (topic_id) REFERENCES
       topic (id)
69 );
```

2.1 Реализация запросов, генерации, индексов

Для генерации данных был реализован собственный скрипт на `Python 3` с использованием библиотеки `MySQL Connector`, который генерирует 1200 записей в таблицу `topic`.

Листинг генерации данных:

```
import mysql.connector

db_config = {
    'host': 'localhost',
    'user': 'root',
    'password': '',
    'database': 'conference_db_lab1',
    'charset': 'utf8mb4'
}

conn = mysql.connector.connect(**db_config)
cursor = conn.cursor()

universities = ['Университет ИТ', 'Технический университет',
   'Государственный университет']

for uni_name in universities:
    cursor.execute("INSERT INTO university (name) VALUES (%s)
       ", (uni_name,))
conn.commit()

number = 1
for university_id in range(1, 4):
    for _ in range(5):
        cursor.execute("INSERT INTO faculty (university_id,
           number) VALUES (%s, %s)", (university_id, number))
        number += 1
conn.commit()

for faculty_id in range(1, 16):
    cursor.execute("SELECT number FROM faculty WHERE id = %s
       ", (faculty_id,))
    faculty_number = cursor.fetchone()[0]
    for group_number in range(1, 5):
        group_name = f'Группа {faculty_number}-{group_number
```

```
            }'
32          cursor.execute("INSERT INTO uni_group (faculty_id,
              name) VALUES (%s, %s)", (faculty_id, group_name))
33  conn.commit()
34
35  for group_id in range(1, 61):
36      for student_num in range(1, 21):
37          student_name = f'Студент {student_num} группы {
              group_id}'
38          cursor.execute("INSERT INTO student (group_id, name)
              VALUES (%s, %s)", (group_id, student_name))
39  conn.commit()
40
41  conferences = [
42      ('Конференция по информатике', 'ГУАП', 'Информатика'),
43      ('Конференция по математике', 'СПбГУ', 'Математика'),
44      ('Конференция по физике', 'ЛГУ', 'Физика')
45  ]
46
47  for conf in conferences:
48      cursor.execute("INSERT INTO conference (name, place,
          theme) VALUES (%s, %s, %s)", conf)
49  conn.commit()
50
51  session_times = [
52      ('09:00:00', '10:30:00'),
53      ('10:45:00', '12:15:00'),
54      ('13:15:00', '14:45:00'),
55      ('15:00:00', '16:30:00'),
56      ('16:45:00', '18:15:00')
57  ]
58
59  conference_dates = {
60      1: '2023-11-01',
61      2: '2023-12-05',
62      3: '2024-01-10'
63  }
64
65  for conference_id in range(1, 4):
66      date = conference_dates[conference_id]
67      for start_time, end_time in session_times:
```

```
68          cursor.execute("""
69              INSERT INTO conference_session (conference_id,
                    start_time, end_time, date)
70              VALUES (%s, %s, %s, %s)
71          """, (conference_id, start_time, end_time, date))
72  conn.commit()
73
74  for session_id in range(1, 16):
75      for topic_num in range(1, 11):
76          title = f'Доклад {topic_num} сессии {session_id}'
77          cursor.execute("INSERT INTO topic (title, session_id)
                VALUES (%s, %s)", (title, session_id))
78  conn.commit()
79
80  author_id = 1
81  for topic_id in range(1, 1201):
82      for _ in range(2):
83          cursor.execute("INSERT INTO authorship (author_id,
                topic_id) VALUES (%s, %s)", (author_id, topic_id))
84          author_id += 1
85          if author_id > 1200:
86              author_id = 1
87  conn.commit()
88
89  cursor.close()
90  conn.close()
91
92  print("Данные успешно сгенерированы и добавлены в базу данных
        .")
```

Для устойчивых замеров для начала сбрасываем кеш БД.

Листинг очистки кеша:

```
1  FLUSH TABLES;
```

На таблице `faculty` создан уникальный индекс `idx_faculty_number` для столбца number с целью предотвращения дублирования данных. Также добавлены индексы для ускорения поиска на различных колонках таблиц `conference`, `uni_group`, `student`, `authorship`, `topic`, `conference_session`, и `faculty`.

Листинг создания индексов:

```
1  USE conference_db_lab1;
```

```
2
3  ALTER TABLE faculty
4    ADD UNIQUE INDEX idx_faculty_number (number);
5
6  CREATE INDEX idx_conference_name ON conference(name);
7  CREATE INDEX idx_conference_id ON conference(id);
8
9  CREATE INDEX idx_uni_group_faculty_id ON uni_group(faculty_id
       );
10 CREATE INDEX idx_student_group_id ON student(group_id);
11 CREATE INDEX idx_authorship_author_id ON authorship(author_id
       );
12 CREATE INDEX idx_authorship_topic_id ON authorship(topic_id);
13 CREATE INDEX idx_topic_session_id ON topic(session_id);
14 CREATE INDEX idx_conference_session_conference_id ON
       conference_session(conference_id);
15
16 CREATE INDEX idx_conference_session_date_time ON
       conference_session(date, start_time);
17
18 CREATE INDEX idx_student_name ON student(name);
19
20 CREATE INDEX idx_faculty_university_id ON faculty(
       university_id);
```

Обычный запрос, взят из предыдущей лабораторной работы. Нужен для меры отсчета производительности.

Листинг обычной проверки быстродействия:

```
1  -- количество докладов для каждой конференции;
2  USE conference_db_lab1;
3
4  SET profiling = 1;
5
6  EXPLAIN ANALYZE
7  SELECT c.id                AS conference_id,
8         COUNT(a.topic_id) AS report_count
9    FROM conference c
10           JOIN
11           conference_session cs ON c.id = cs.conference_id
12           JOIN
13           topic t ON cs.id = t.session_id
```

```
14          JOIN
15          authorship a ON t.id = a.topic_id
16    GROUP BY c.id;
17
18  SHOW PROFILES;
```

Выполнен SQL-запрос, использующий соединение нескольких таблиц с целью подсчета количества докладов для каждой конференции. С использованием `EXPLAIN ANALYZE` проведен анализ производительности запроса. Прямо указано использование индекса `idx_conference_session_conference_id`, что помогает оптимизатору базы данных выбрать конкретный план исполнения запроса.

Листинг проверки быстродействия с явным указанием индекса:

```
1   -- количество докладов для каждой конференции;
2   USE conference_db_lab1;
3
4   EXPLAIN ANALYZE
5   SELECT
6     c.id AS conference_id,
7     COUNT(a.topic_id) AS report_count
8     FROM conference c
9           JOIN conference_session cs USE INDEX (
              idx_conference_session_conference_id) ON c.id =
              cs.conference_id
10          JOIN topic t ON cs.id = t.session_id
11          JOIN authorship a ON t.id = a.topic_id
12    GROUP BY c.id;
```

Представлен аналогичный запрос без явного указания индексов, чтобы сравнить, как отсутствие индексов влияет на производительность запроса.

Листинг проверки быстродействия без использования индексов:

```
1   -- количество докладов для каждой конференции;
2   USE conference_db_lab1;
3
4   EXPLAIN ANALYZE
5   SELECT
6     c.id AS conference_id,
7     COUNT(a.topic_id) AS report_count
8     FROM conference c
9           JOIN conference_session cs USE INDEX (
```

```
            idx_conference_session_conference_id) ON c.id =
            cs.conference_id
10        JOIN topic t ON cs.id = t.session_id
11        JOIN authorship a ON t.id = a.topic_id
12  GROUP BY c.id;
```

2.2 Проверка результатов

2.2.1 Без явного указания стратегии

Для начала выполним скрипт без явного указания стратегии анализатору. Получим следующее.

```
1  -> Group aggregate: count(a.topic_id)  (cost=119 rows=3) (
     actual time=0.102..0.242 rows=3 loops=1)
2      -> Nested loop inner join  (cost=89.1 rows=300) (actual
       time=0.0333..0.233 rows=300 loops=1)
3          -> Nested loop inner join  (cost=21.6 rows=150) (
           actual time=0.0281..0.0664 rows=150 loops=1)
4              -> Nested loop inner join  (cost=2.8 rows=15) (
               actual time=0.0248..0.0291 rows=15 loops=1)
5                  -> Covering index scan on c using
                   idx_conference_id  (cost=0.55 rows=3) (
                   actual time=0.0168..0.0175 rows=3 loops=1)
6                  -> Covering index lookup on cs using
                   idx_conference_session_conference_id (
                   conference_id=c.id)  (cost=0.418 rows=5) (
                   actual time=0.00306..0.00354 rows=5 loops
                   =3)
7              -> Covering index lookup on t using
               idx_topic_session_id (session_id=cs.id)  (cost
               =0.319 rows=10) (actual time=0.00129..0.00205
               rows=10 loops=15)
8          -> Covering index lookup on a using
           idx_authorship_topic_id (topic_id=t.id)  (cost
           =0.252 rows=2) (actual time=743e-6..971e-6 rows=2
           loops=150)
```

Индекс idx_conference_id обеспечивает сканирование с возвратом 3 строк за 0.0168 до 0.0175 миллисекунд. Далее, через индекс idx_conference_session_conference_id, выполняется поиск по conference_id в conference_session, с возвратом 5 строк, за каждые 3 итерации цикла, в среднем за 0.00306 до 0.00354 мс. Индекс idx_topic_session_id ускоряет поиск

по session_id, возвращая 10 строк за 0.00129 до 0.00205 мс на 15 итерациях. Наконец, с помощью индекса idx_authorship_topic_id, извлекаются данные по topic_id с двумя строками почти за микросекунды.

2.2.2 С явным указанием индекса

При явном указании индекса получаем следующее:

```
1 -> Group aggregate: count(a.topic_id) (cost=119 rows=3) (
  actual time=0.121..0.251 rows=3 loops=1)
2     -> Nested loop inner join (cost=89.1 rows=300) (actual
      time=0.0464..0.242 rows=300 loops=1)
3         -> Nested loop inner join (cost=21.6 rows=150) (
          actual time=0.0403..0.0779 rows=150 loops=1)
4             -> Nested loop inner join (cost=2.8 rows=15) (
              actual time=0.0358..0.0402 rows=15 loops=1)
5                 -> Covering index scan on c using
                  idx_conference_id (cost=0.55 rows=3) (
                  actual time=0.0252..0.026 rows=3 loops=1)
6                 -> Covering index lookup on cs using
                  idx_conference_session_conference_id (
                  conference_id=c.id) (cost=0.418 rows=5) (
                  actual time=0.00394..0.00438 rows=5 loops
                  =3)
7             -> Covering index lookup on t using
              idx_topic_session_id (session_id=cs.id) (cost
              =0.319 rows=10) (actual time=0.00139..0.0021
              rows=10 loops=15)
8         -> Covering index lookup on a using
          idx_authorship_topic_id (topic_id=t.id) (cost
          =0.252 rows=2) (actual time=747e-6..962e-6 rows=2
          loops=150)
```

Запрос использует индексы для повышения эффективности: сначала, сканирование по индексу idx_conference_id возвращает 3 строки за 0.0252 до 0.026 миллисекунд. Затем, индекс idx_conference_session_conference_id позволяет выполнить поиск по conference_id в conference_session, возвращая 5 строк в среднем за 0.00394 до 0.00438 миллисекунд на 3 итерации. Индекс idx_topic_session_id оптимизирует поиск по session_id, возвращая 10 строк за 0.00139 до 0.0021 миллисекунд на 15 итерациях. Наконец, индекс idx_authorship_topic_id использован для извлечения данных по topic_id, возвращая две строки почти мгновенно — от 747 до 962 микросекунд на 150

итерациях, что иллюстрирует общее ускорение выполнения запроса за счет использования индексирования.

2.2.3 Явный запрет использования индексов

Получаем следующее:

```
1  -> Table scan on <temporary>  (actual time=0.367..0.367 rows
       =3 loops=1)
2      -> Aggregate using temporary table  (actual time
          =0.366..0.366 rows=3 loops=1)
3          -> Inner hash join (a.topic_id = t.id)  (cost=4731
             rows=300) (actual time=0.259..0.311 rows=300 loops
             =1)
4              -> Table scan on a  (cost=0.00302 rows=300) (
                 actual time=0.0103..0.038 rows=300 loops=1)
5              -> Hash
6                  -> Inner hash join (t.session_id = cs.id)  (
                     cost=231 rows=150) (actual time
                     =0.12..0.182 rows=150 loops=1)
7                      -> Table scan on t  (cost=0.0833 rows
                         =150) (actual time=0.0242..0.0555 rows
                         =150 loops=1)
8                      -> Hash
9                          -> Inner hash join (cs.conference_id
                             = c.id)  (cost=5.3 rows=15) (
                             actual time=0.0755..0.0817 rows=15
                             loops=1)
10                             -> Table scan on cs  (cost=0.25
                                 rows=15) (actual time
                                 =0.00813..0.011 rows=15 loops
                                 =1)
11                             -> Hash
12                                 -> Table scan on c  (cost
                                     =0.55 rows=3) (actual time
                                     =0.0484..0.0528 rows=3
                                     loops=1)
```

Запрос выполняет последовательное сканирование таблиц и использует хеш-соединения для агрегации данных. Сначала сканируется временная таблица за 0.367 миллисекунд, возвращая 3 строки, с помощью временной таблицы выполняется агрегация за 0.366 миллисекунд. Хеш-соединение между таблицами a и t обрабатывает 300 строк за 0.259–0.311 миллисекунд,

с полным сканированием a за 0.0103–0.038 миллисекунд. Для t и cs хеш-соединение и сканирование t обрабатывают 150 строк за 0.0242–0.0555 миллисекунд. Наконец, хеш-соединение cs и c с полным сканированием cs занимает 0.00813–0.011 миллисекунд, а хеширование c — 0.0484–0.0528 миллисекунд. Как мы видим, быстродействие запроса заметно деградировало, при явном запрете использования индексов.

3 Вывод

В ходе выполнения лабораторной работы было изучено понятие индексации данных и ей назначение. Реализованы индексы при выполнении запросов в собственной базе данных и проанализированы полученные результаты. Как мы увидели, самый быстрый результат был получен, когда анализатор сам выбирал стратегию выполнения, а самый худший - при запрете использования индексов. Разница составила почти 40%.

ПРИЛОЖЕНИЕ

```
import mysql.connector

db_config = {
    'host': 'localhost',
    'user': 'root',
    'password': '',
    'database': 'conference_db_lab1',
    'charset': 'utf8mb4'
}

conn = mysql.connector.connect(**db_config)
cursor = conn.cursor()

universities = ['Университет ИТ', 'Технический университет',
   'Государственный университет']

for uni_name in universities:
    cursor.execute("INSERT INTO university (name) VALUES (%s)
       ", (uni_name,))
conn.commit()

number = 1
for university_id in range(1, 4):
    for _ in range(5):
        cursor.execute("INSERT INTO faculty (university_id,
           number) VALUES (%s, %s)", (university_id, number))
        number += 1
conn.commit()

for faculty_id in range(1, 16):
    cursor.execute("SELECT number FROM faculty WHERE id = %s
       ", (faculty_id,))
    faculty_number = cursor.fetchone()[0]
    for group_number in range(1, 5):
        group_name = f'Группа {faculty_number}-{group_number
           }'
        cursor.execute("INSERT INTO uni_group (faculty_id,
           name) VALUES (%s, %s)", (faculty_id, group_name))
conn.commit()

```

```
35 for group_id in range(1, 61):
36     for student_num in range(1, 21):
37         student_name = f'Студент {student_num} группы {
               group_id}'
38         cursor.execute("INSERT INTO student (group_id, name)
               VALUES (%s, %s)", (group_id, student_name))
39 conn.commit()
40
41 conferences = [
42     ('Конференция по информатике', 'ГУАП', 'Информатика'),
43     ('Конференция по математике', 'СПбГУ', 'Математика'),
44     ('Конференция по физике', 'ЛГУ', 'Физика')
45 ]
46
47 for conf in conferences:
48     cursor.execute("INSERT INTO conference (name, place,
           theme) VALUES (%s, %s, %s)", conf)
49 conn.commit()
50
51 session_times = [
52     ('09:00:00', '10:30:00'),
53     ('10:45:00', '12:15:00'),
54     ('13:15:00', '14:45:00'),
55     ('15:00:00', '16:30:00'),
56     ('16:45:00', '18:15:00')
57 ]
58
59 conference_dates = {
60     1: '2023-11-01',
61     2: '2023-12-05',
62     3: '2024-01-10'
63 }
64
65 for conference_id in range(1, 4):
66     date = conference_dates[conference_id]
67     for start_time, end_time in session_times:
68         cursor.execute("""
69             INSERT INTO conference_session (conference_id,
                   start_time, end_time, date)
70             VALUES (%s, %s, %s, %s)
71         """, (conference_id, start_time, end_time, date))
```

```
conn.commit()

for session_id in range(1, 16):
    for topic_num in range(1, 11):
        title = f'Доклад {topic_num} сессии {session_id}'
        cursor.execute("INSERT INTO topic (title, session_id)
            VALUES (%s, %s)", (title, session_id))
conn.commit()

author_id = 1
for topic_id in range(1, 151):
    for _ in range(2):
        cursor.execute("INSERT INTO authorship (author_id,
           topic_id) VALUES (%s, %s)", (author_id, topic_id))
        author_id += 1
        if author_id > 1200:
            author_id = 1
conn.commit()

cursor.close()
conn.close()

print("Данные успешно сгенерированы и добавлены в базу данных
   .")
```

```
USE conference_db_lab1;

ALTER TABLE faculty
  ADD UNIQUE INDEX idx_faculty_number (number);

CREATE INDEX idx_conference_name ON conference(name);
CREATE INDEX idx_conference_id ON conference(id);

CREATE INDEX idx_uni_group_faculty_id ON uni_group(faculty_id
   );
CREATE INDEX idx_student_group_id ON student(group_id);
CREATE INDEX idx_authorship_author_id ON authorship(author_id
   );
CREATE INDEX idx_authorship_topic_id ON authorship(topic_id);
CREATE INDEX idx_topic_session_id ON topic(session_id);
CREATE INDEX idx_conference_session_conference_id ON
   conference_session(conference_id);
```

```
15
16 CREATE INDEX idx_conference_session_date_time ON
       conference_session(date, start_time);
17
18 CREATE INDEX idx_student_name ON student(name);
19
20 CREATE INDEX idx_faculty_university_id ON faculty(
       university_id);
```

```
1  -- количество докладов для каждой конференции;
2  USE conference_db_lab1;
3
4  SET profiling = 1;
5
6  EXPLAIN ANALYZE
7  SELECT c.id               AS conference_id,
8         COUNT(a.topic_id) AS report_count
9    FROM conference c
10          JOIN
11          conference_session cs ON c.id = cs.conference_id
12          JOIN
13          topic t ON cs.id = t.session_id
14          JOIN
15          authorship a ON t.id = a.topic_id
16   GROUP BY c.id;
17
18 SHOW PROFILES;
```

```
1  -- количество докладов для каждой конференции;
2  USE conference_db_lab1;
3
4  EXPLAIN ANALYZE
5  SELECT
6    c.id AS conference_id,
7    COUNT(a.topic_id) AS report_count
8    FROM conference c
9           JOIN conference_session cs USE INDEX (
              idx_conference_session_conference_id) ON c.id =
              cs.conference_id
10          JOIN topic t ON cs.id = t.session_id
11          JOIN authorship a ON t.id = a.topic_id
12   GROUP BY c.id;
```

```sql
-- количество докладов для каждой конференции;
USE conference_db_lab1;

EXPLAIN ANALYZE
SELECT
  c.id AS conference_id,
  COUNT(a.topic_id) AS report_count
  FROM conference c
        JOIN conference_session cs USE INDEX (
           idx_conference_session_conference_id) ON c.id =
           cs.conference_id
        JOIN topic t ON cs.id = t.session_id
        JOIN authorship a ON t.id = a.topic_id
  GROUP BY c.id;
```