**ЛАБОРАТОРНАЯ РАБОТА №5**

**«ИССЛЕДОВАНИЕ ПРИНЦИПОВ ПАРТИЦИРОВАНИЯ ДАННЫХ»**

**Цель работы**

Исследовать способы партицирования таблиц баз данных и их влияние на скорость доступа с данным. Изучить основы партицирования на примере MySQL.

**Вариант задания**

|  |  |  |
| --- | --- | --- |
| Вариант | Название таблицы (в скобках  указан перечень полей) | Кол-во строк в таблице |
| 1 | articles (id, title, price) | 100, 500, 1000, 2000 |

**Ход выполнения работы**

Был сформирован файл для создания базы данных. Его содержимое представлено в листинге 1.

Листинг 1 - Файл database.sql

--

-- Скрипт создания базы данных с данными.

--

-- Удалить старую базу и аккаунт, если они существуют

DROP DATABASE IF EXISTS lab7;

DROP USER IF EXISTS lab7User;

-- Создать новую базу и аккаунт

CREATE DATABASE lab7;

CREATE USER lab7User IDENTIFIED BY 'lab7Password';

GRANT ALL PRIVILEGES ON lab7.\* TO lab7User;

-- Использовать созданную базу

USE lab7;

-- Создать таблицу без партицирования

CREATE TABLE IF NOT EXISTS `blogs\_no\_part` (

    `id` INTEGER NOT NULL AUTO\_INCREMENT,

    `title` VARCHAR(256) NOT NULL,

    `posts\_count` INTEGER NOT NULL DEFAULT 0,

    PRIMARY KEY (`id`)

) ENGINE = InnoDB;

-- Создать таблицу с партицированием

CREATE TABLE IF NOT EXISTS `blogs\_part` (

    `id` INTEGER NOT NULL AUTO\_INCREMENT,

    `title` VARCHAR(256) NOT NULL,

    `posts\_count` INTEGER NOT NULL DEFAULT 0,

    PRIMARY KEY (`id`, `posts\_count`)

) ENGINE = InnoDB

  PARTITION BY RANGE(`posts\_count`) (

     PARTITION p0 VALUES LESS THAN (10),

     PARTITION p1 VALUES LESS THAN (20),

     PARTITION p2 VALUES LESS THAN (30),

     PARTITION p3 VALUES LESS THAN (40),

     PARTITION p4 VALUES LESS THAN MAXVALUE);

-- Создать процедуру для генерации случайных данных

DELIMITER //

CREATE PROCEDURE `generate\_data` (IN items\_count INT)

BEGIN

    DECLARE i INT DEFAULT 0;

    WHILE i < items\_count DO

        INSERT INTO `blogs\_no\_part` (`title`, `posts\_count`) VALUES (SUBSTRING(MD5(RAND()) FROM 1 FOR 8), RAND() \* 1000);

        SET i = i + 1;

    END WHILE;

    -- Скопировать те же данные в партицированную таблицу

    INSERT INTO `blogs\_part`

        SELECT \* FROM `blogs\_no\_part`;

END//

DELIMITER ;

-- Заполнить таблицу данными

CALL generate\_data(1900);

-- Удалить процедуру

DROP PROCEDURE `generate\_data`;

Для формирования страницы с результатами был создан файл программы nodejs. Его содержимое представлено в листинге 5.3.

Листинг 5.3. Файл server.js

const express = require('express');

const mysql = require('mysql2/promise');

const hirestime = require('hirestime')

'use strict';

// Параметры web-сервера

const server\_host = 'localhost';

const server\_port = '3000';

// Параметры подключения к MySQL

const conn = { host: 'localhost', port: 3305, database: 'lab7', user: 'lab7User', password : 'lab7Password' };

// Номер id для оценки времени

const test\_id = 100;

// Тестовые запросы

const queries = [

    { table: 'blogs\_no\_part', sql: `SELECT SQL\_NO\_CACHE \* FROM blogs\_no\_part LIMIT 1` },

    { table: 'blogs\_no\_part', sql: `SELECT SQL\_NO\_CACHE \* FROM blogs\_no\_part WHERE id = ${test\_id} LIMIT 1` },

    { table: 'blogs\_part',    sql: `SELECT SQL\_NO\_CACHE \* FROM blogs\_part LIMIT 1` },

    { table: 'blogs\_part',    sql: `SELECT SQL\_NO\_CACHE \* FROM blogs\_part WHERE id = ${test\_id} LIMIT 1` },

];

async function run() {

    // Установить соединение с сервером

    const db = await mysql.createConnection(conn);

    // Создать web сервис

    const app = express();

    // Использовать pug (бывший jade) как HTML template движок

    app.set('view engine', 'pug');

    // По запросу http://server:port/ выполнить 4 тестовых выборки из двух таблиц

    app.get('/', wrapAsync(async (req, res, next) => {

    // Получить количество элементов в таблице (только для отчета)

    const [ rows ] = await db.execute(`SELECT COUNT(\*) AS n FROM ${queries[0].table}`);

    const n = rows[0].n;

    // Выполнить тестовые выборки, накапливая результаты в массиве results

    let results = [];

    for (let i in queries) {

        let time = hirestime();

        let [ rows, fields ] = await db.execute(queries[i].sql); // Добавлено получение fields

        let elapsed = time(hirestime.MS).toFixed(2);

        results.push({

            n: n,

            sql: queries[i].sql,

            table: queries[i].table,

            time: elapsed,

            rows: rows, // Добавлено сохранение результатов запроса

            fields: fields // Добавлено сохранение метаданных о полях

        });

    }

    // Отобразить страницу результатов с помощью pug по шаблону results.pug

    res.render('results', { header: `Query results`, results: results });

}));

    // Запустить HTTP сервер

    app.listen(server\_port, server\_host, () => {

        console.log(`Server running at http://${server\_host}:${server\_port}/, press Ctrl-C to exit`);

    });

}

// Вспомогательная функция для обработки возможных ошибок в асинхронных функциях.

// Переназначает exceptions на стандартный обработчик ошибок Express.

function wrapAsync(fn) {

    return function(req, res, next) {

        fn(req, res, next).catch(next);

    };

}

run();

При помощи модуля pug был создан файл шаблона, а также описаны стили. Это представлено в листингах 5.4-5.5.

Листинг 5.4. Файл results.pug

html

  head

    title= 'Lab5'

    style

      include style.css

  body

    h2= header

    table

      tr

        th= 'Records'

        th= 'SQL query'

        th= 'SQL table'

        th= 'Time (ms)'

        for res in results

          tr

            td= res.n

            td= res.sql

            td= res.table

            td= res.time

Листинг 5.5. Файл style.css

table {

    border-collapse: collapse;

}

th, td {

    border: 1px solid blue;

    padding: 5px;

}

Результаты работы запросов к таблицам без и с использованием партицирования представлены на рисунках 1 – 5.

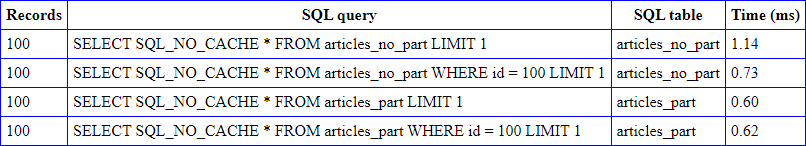


Рисунок 1 – Время выполнения запросов для таблицы с 100 записями

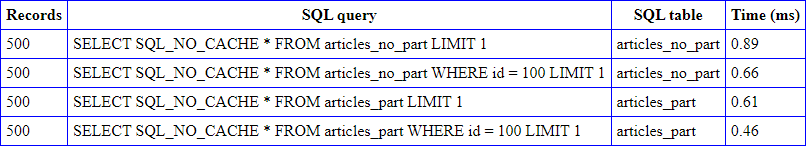


Рисунок 2 – Время выполнения запросов для таблицы с 500 записями

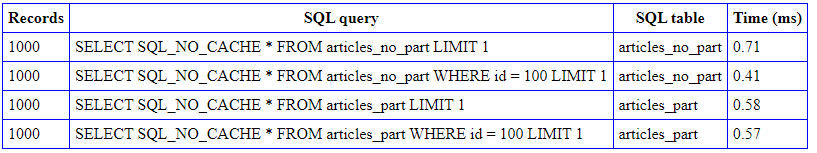


Рисунок 3 – Время выполнения запросов для таблицы с 1000 записями

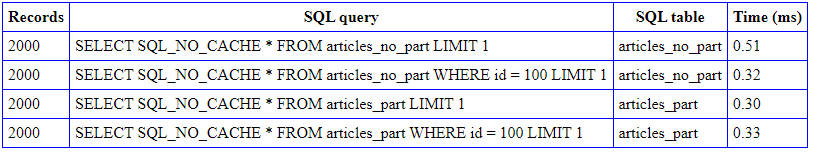


Рисунок 4 – Время выполнения запросов для таблицы с 2000 записями

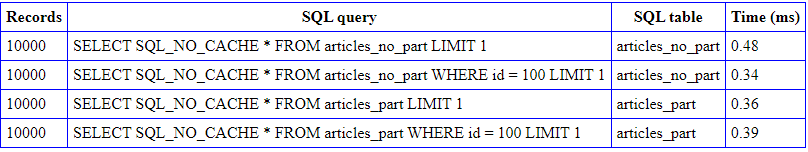


Рисунок 5 – Время выполнения запросов для таблицы с 10000 записями

**Вывод**

Таким образом было выявлено, что партицирование позволяет сократить время выполнения запросов к базе данных при условии правильного разбиения таблиц на партиции.