# Лабораторная «Исследование принципов партицирования баз данных»

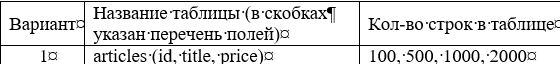
## Цель работы

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

## Постановка задачи

Используя методику, описанную в приложении, создать простой HTTP

+ MySQL сервер и реализовать партицирование. Вариант таблицы и число строк для запуска скрипта приведены ниже:

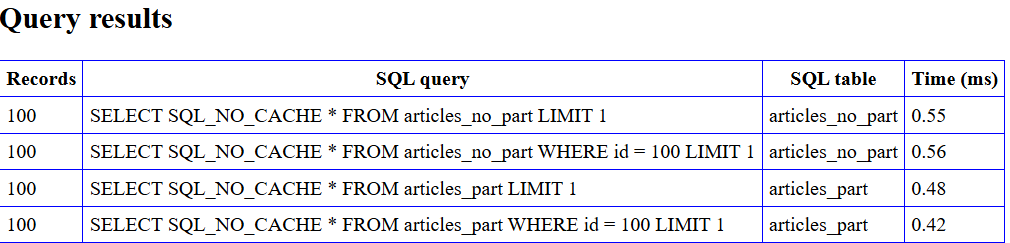


* + - * 1. Вариант задания

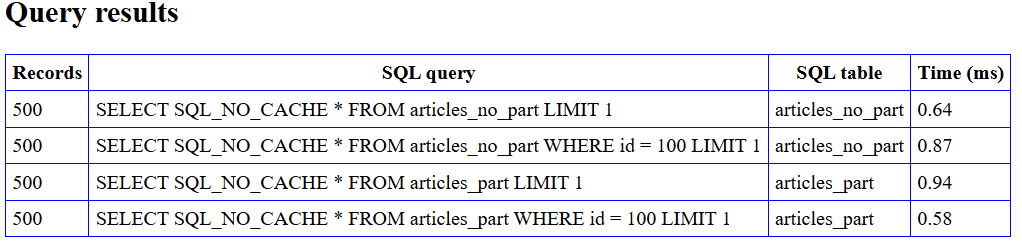
## Ход работы

В ходе работы были созданы две таблицы в базе данных article: первая таблица articles без партиционирования и вторая articles\_part, в которой данные разделены на партиции по значению price. После создания структуры базы данных была разработана программа на JavaScript для генерации данных. В обе таблицы было добавлено 2000 записей, каждая из которых содержала уникальное название и одно из четырех значений цены: 100, 500, 1000, 2000.

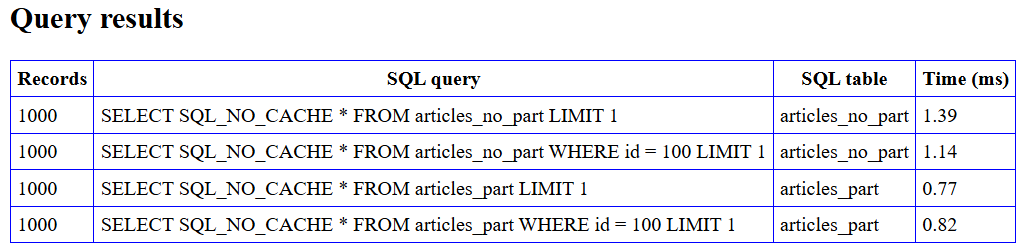
Для оценки влияния партиционирования на производительность были проведены измерения времени выполнения запросов при различном количестве записей в таблице. На рисунке 1.2 показано время выполнения запросов при 100 записях, на рисунке 1.3 приведены результаты для 500 записей, на рисунке 1.4 представлено время выполнения при 1000 записях, а на рисунке 1.5 – при 2000 записях.



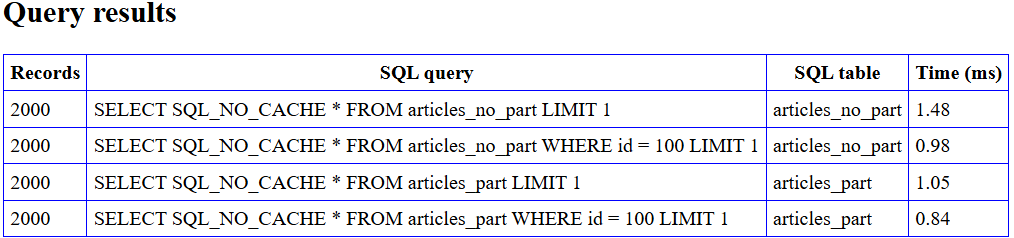
* + - * 1. Время запросов при 100 записях



* + - * 1. Время запросов при 500 записях

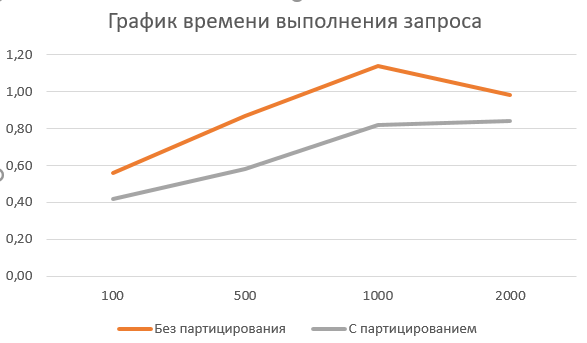


* + - * 1. Время запроса при 1000 записях



* + - * 1. Время запроса при 2000 записях

На рисунке 1.6 показан график зависимости времени выполнения запроса от количества записей.



* + - * 1. График зависимости время выполнения запроса от количества записей

В листинге 1.1 показан код программы.

Код файла 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: '127.0.0.1',

    user: 'root',

    password: 'кщще',

    database: 'uir5'

};

const test\_id = 100;

// 🔹 Тестовые запросы

const queries = [

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

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

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

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

];

async function run() {

    // Подключение к базе данных

    const db = await mysql.createConnection(conn);

    // Создание web-сервера

    const app = express();

    // Использование pug как шаблонизатора

    app.set('view engine', 'pug');

    // Обработчик запроса на главную страницу

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

        // Подсчет количества записей в таблице (только для отчета)

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

        const n = rows[0].n;

        // Выполнение тестовых выборок и измерение времени выполнения

        let results = [];

        for (let i in queries) {

            let time = hirestime();

            let [ rows ] = await db.execute(queries[i].sql);

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

            results.push({

                n: n,

                sql: queries[i].sql,

                table: queries[i].table,

                time: elapsed,

            });

        }

        // Отображение страницы с результатами

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

    }));

    // Запуск сервера

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

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

    });

}

// 🔹 Функция-обёртка для обработки ошибок в асинхронных функциях

function wrapAsync(fn) {

    return function(req, res, next) {

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

    };

}

// 🔹 Запуск сервера

run();

Код генерации данных показан в листинге 1.2.

Генерация данных

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

const dbConfig = {

    host: '127.0.0.1',

    user: 'root',

    password: 'кщще',

    database: 'uir5'

};

async function generateData() {

    const connection = await mysql.createConnection(dbConfig);

    try {

        console.log("Подключение к базе данных...");

        // Очищаем таблицы перед вставкой

        await connection.execute('DELETE FROM articles\_no\_part');

        await connection.execute('DELETE FROM articles\_part');

        console.log("Таблицы очищены. Начинаем генерацию данных...");

        let values = [];

        for (let i = 1; i <= 500; i++) {

            let price;

            switch (i % 4) {

                case 0: price = 100; break;

                case 1: price = 500; break;

                case 2: price = 1000; break;

                default: price = 2000; break;

            }

            values.push([`Article ${i}`, price]);

        }

        // SQL-запрос для вставки данных в articles\_no\_part

        const sqlNoPart = "INSERT INTO articles\_no\_part (title, price) VALUES ?";

        await connection.query(sqlNoPart, [values]);

        // Добавляем ID для партиционированной таблицы

        const valuesWithId = values.map((val, index) => [index + 1, ...val]);

        // SQL-запрос для вставки данных в articles\_part

        const sqlPart = "INSERT INTO articles\_part (id, title, price) VALUES ?";

        await connection.query(sqlPart, [valuesWithId]);

        console.log("Генерация завершена! 2000 записей добавлено в обе таблицы.");

    } catch (error) {

        console.error("Ошибка при генерации данных:", error);

    } finally {

        await connection.end();

    }

}

// Запускаем генерацию данных

generateData();

## Вывод

В ходе лабораторной работы были изучены принципы партиционирования баз данных и проведено сравнение производительности запросов к обычной и партиционированной таблице. Были разработаны SQL-скрипты для создания таблиц, программа на JavaScript для генерации тестовых данных, а также Node.js сервер, выполняющий тестовые запросы и измеряющий их время выполнения.

Проведенные измерения подтвердили, что при небольших объемах данных разница в скорости выполнения запросов практически незаметна. Однако с увеличением количества записей использование партиционирования значительно улучшает производительность, что особенно важно для обработки больших объемов информации. Таким образом, применение партиционирования в базах данных может быть полезным инструментом для оптимизации работы с большими данными и ускорения выполнения запросов.