# ЛАБОРАТОРНАЯ РАБОТА №1

# «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

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

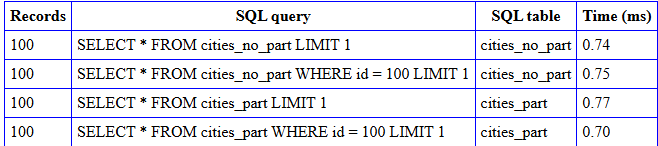
Таблица 1.1 – Вариант задания

|  |  |  |
| --- | --- | --- |
| Вариант | Название таблицы (в скобках  указан перечень полей) | Кол-во строк в таблице |
| 3 | cities (id, title, short\_code) | 300, 800, 1400, 1700 |

## Ход работы

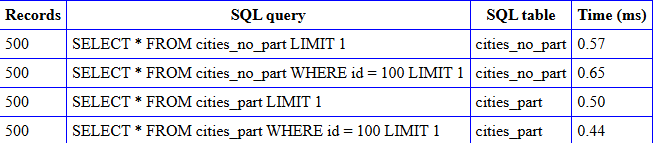
В ходе выполнения работы бsли созданы две таблицы в базе данных cities: таблица без использования партицирования и таблица в которой данные разделены на партиции по id. После создания базы данных, была разработана программа JS для генерации данных. В каждую таблицу было добавлено 100, 500, 1000 и 2000 щаписей.

Для оценки эффективности партицирования и его влияния на производительность провели измерения времени выполнения запросов при различном количестве записей в таблицах. На рисунках 1.1 – 1.4 показан результат при разном количестве записей.

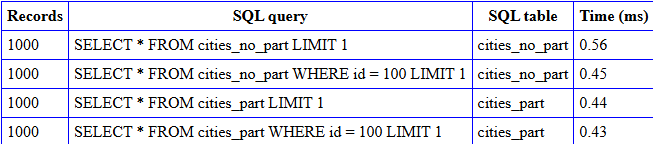


* + - * 1. Время запросов при 100 записях

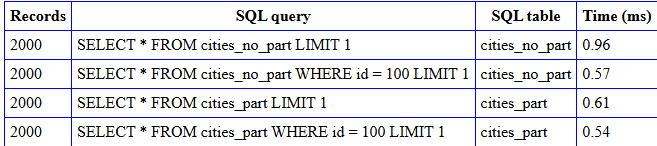
Как видно на рисунке 1.1 влияние пртицирования не сильно заметно при малом объёме данных.



* + - * 1. Время запросов при 500 записях

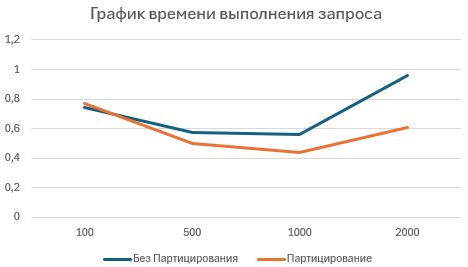


* + - * 1. Время запросов при 1000 записях



* + - * 1. Время запросов при 2000 записях

На рисунке 1.5 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 { Pool } = require('pg');

const hirestime = require('hirestime');

'use strict';

// 🔹 Параметры web-сервера

const server\_host = 'localhost';

const server\_port = '3000';

// 🔹 Параметры подключения к PostgreSQL

const conn = {

    host: '127.0.0.1',

    user: 'postgres',

    password: '7676',

    database: 'uir5',

    port: 5432, // стандартный порт PostgreSQL

};

const test\_id = 100;

// 🔹 Тестовые запросы

const queries = [

    { table: 'cities\_no\_part', sql: `SELECT \* FROM cities\_no\_part LIMIT 1` },

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

    { table: 'cities\_part', sql: `SELECT \* FROM cities\_part LIMIT 1` },

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

];

async function run() {

    // Подключение к базе данных

    const pool = new Pool(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 pool.query(`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 pool.query(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 { Pool } = require('pg');

const dbConfig = {

    host: '127.0.0.1',

    user: 'postgres',

    password: '7676',

    database: 'uir5',

    port: 5432

};

const N = 2000; // Количество записей для генерации

async function generateData() {

    const pool = new Pool(dbConfig);

    const client = await pool.connect();

    try {

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

        // Очищаем таблицы перед вставкой

        await client.query('DELETE FROM cities\_no\_part');

        await client.query('DELETE FROM cities\_part');

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

        // Вставка в cities\_no\_part

        let values = [];

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

            values.push([`City ${i}`, `C${i}`]);

        }

        const insertNoPart = `

            INSERT INTO cities\_no\_part (title, short\_code)

            VALUES ${values.map((\_, i) => `($${i \* 2 + 1}, $${i \* 2 + 2})`).join(', ')}

        `;

        await client.query(insertNoPart, values.flat());

        // Вставка в cities\_part (с id)

        let valuesWithId = [];

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

            valuesWithId.push([i, `City ${i}`, `C${i}`]);

        }

        const insertPart = `

            INSERT INTO cities\_part (id, title, short\_code)

            VALUES ${valuesWithId.map((\_, i) => `($${i \* 3 + 1}, $${i \* 3 + 2}, $${i \* 3 + 3})`).join(', ')}

        `;

        await client.query(insertPart, valuesWithId.flat());

        console.log(`Генерация завершена! ${N} записей добавлено в обе таблицы.`);

    } catch (error) {

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

    } finally {

        client.release();

        await pool.end();

    }

}

// Запускаем генерацию данных

generateData();

## Вывод