1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 4**

**Физическая организация и производительность операций с данными**

по дисциплине «Системы управления базами данных»

1. Выполнила
2. студентка гр. 4851003/90801 Кулеева А.Г.

1. Руководитель
2. ассистент Зубков Е.А.
3. Санкт-Петербург
4. 2023

# **Цель**

Получение навыков оценки влияния организации данных и метода доступа при поиске на производительность процессов выборки и модификации информации в базе данных.

# Ход работы

## Анализ размера таблицы

Рассмотрим пример выполнения скрипта, выводящего размер базы данных. Функция pg\_database\_size (name/oid) вычисляет общий объём, который занимает на диске база данных с указанным именем или OID. Для использования этой функции необходимо иметь право CONNECT для заданной базы (оно даётся по умолчанию) или быть членом роли pg\_read\_all\_stats. Функция pg\_size\_pretty (bigint) преобразует размер в байтах в более понятный человеку формат с единицами измерения (bytes, kB, MB, GB или TB, в зависимости от значения). На рисунке 1 представлен результат вывода. Таким образом, все базы занимают 23215 Кб = 22 Мб.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 ― Размер баз данных

Требуется создать дополнительную таблицу, в которую будут сохраняться лог-сообщения обо всех операциях, которые совершает тот или иной пользователь в отношении основных таблиц. Создадим для этого новую бд, на которой будем ставить эксперименты.

CREATE DATABASE lab4;

\c lab4

CREATE TABLE MOCKDATA(

first\_name VARCHAR(500),

last\_name VARCHAR(500),

gender VARCHAR(5),

country VARCHAR(500));

Для логирования была создана дополнительная таблица (на основе л.р. 3) и соответствующий ей триггер, срабатывающий при попытке пользователя получить доступ к данным.

drop table if exists logging\_table;

create table logging\_table (

username text,

tablename text not null,

operation text not null,

mtime timestamp not null);

create or replace function log\_update()

returns trigger as $log\_update$

begin

insert into logging\_table (username, tablename, operation, mtime)

values (current\_user, TG\_TABLE\_NAME, TG\_OP, now());

return null;

end;

$log\_update$ language plpgsql;

drop trigger if exists logging\_trigger on MOCKDATA;

create trigger logging\_trigger

after insert or update or delete on MOCKDATA

for each row execute procedure log\_update();

Функция pg\_table\_size(regclass) вычисляет объём, который занимает на диске данная таблица, за исключением индексов (но включая её TOAST-таблицу (если она есть), карту свободного места и карту видимости). Функция pg\_total\_relation\_size(regclass) вычисляет общий объём, который занимает на диске заданная таблица, включая все её индексы и данные TOAST. Результат этой функции равен значению pg\_table\_size + pg\_indexes\_size. На рисунке 2 представлен результат вычисления размеров пустой таблицы логирования.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 ― Размер пустой таблицы

Сгенерируем случайные данные с помощью сервиса генерации данных. Было создано 3 файла размером 10, 100 и 1000 записей. Импортируем данные в таблицу и посмотрим, как изменилась таблица логов (рисунок 3).

\i C:/Users/heath/Downloads/ExportSQL(10).sql

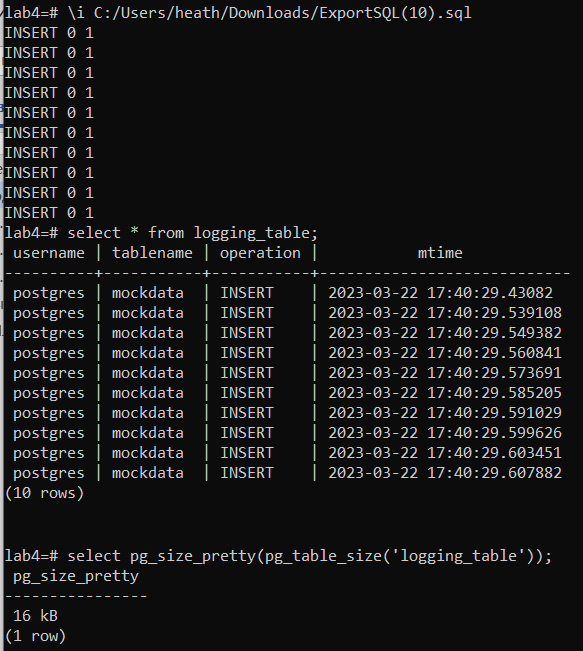


Рисунок 3 ― Изменение размера таблицы при 10 записях

При 100 записях размер таблицы не изменился, при 1000 изменился в 7 раз (рисунок 4).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 ― Изменение при 1000 записях

В таблице 1 представлен результат увеличения базы данных. Как видно, размер одной записи уменьшается пропорционально увеличению размера таблицы. Это связано с тем, что данные хранятся в сжатом виде.

Таблица 1 — Размер на диске при увеличении количества записей

|  |  |  |
| --- | --- | --- |
| Количество записей | Размер на диске | Размер одной записи |
| 0 | 8Кб | - |
| 10 | 16Кб | 819 байт |
| 100 | 16Кб | 81 байт |
| 1000 | 112Кб | 106 байт |
| 2000 | 184Кб | 92 байта |
| 3000 | 248Кб | 81 байт |
| 4000 | 320Кб | 80 байт |

Попробуем подсчитать средний размер записи. Для этого необходимо рассчитать средний размер для каждого измерения, после чего рассчитать среднее значение полученных результатов. Не забываем вычитать 8Кб, которые занимает сама таблица.

Однако есть более точный показатель: среднее гармоническое. Среднее гармоническое — один из способов, которым можно понимать «среднюю» величину некоторого набора чисел. Его можно определить следующим образом: пусть даны положительные числа x1, …, xn,�1,…,�� тогда их средним гармоническим будет такое число H�, что

��=1�1+…+1��Можно получить явную формулу для среднего гармонического:�(�1,…,��)=�1�1+1�2+⋯+1��=�∑�=1�1��

Таким образом, средний размер записи равен 102 байта. Возможно, что размер записи может меняться также в силу сжатия данных.

## Секционирование

Очистим таблицу и сделаем секционирование встроенными средствами СУБД. Рассмотрим вариант декларативного секционирования.

1. Создайте таблицу как секционированную таблицу с предложением PARTITION BY, указав метод разбиения (в нашем случае RANGE) и список столбцов, которые будут образовывать ключ разбиения.

CREATE TABLE MOCKDATA(

first\_name VARCHAR(50),

last\_name VARCHAR(50),

gender VARCHAR(5),

country VARCHAR(50),

age INT

) PARTITION BY RANGE (age);

1. Создайте секции. В определении каждой секции должны задаваться границы, соответствующие методу и ключу разбиения родительской таблицы. Заметьте, что указание границ, при котором множество значений новой секции пересекается со множеством значений в одной или нескольких существующих секциях, будет ошибочным. Секции, создаваемые таким образом, во всех отношениях являются обычными таблицами. В частности, для каждой секции можно независимо задать табличное пространство и параметры хранения.

В данном примере ключом для секционирования является возраст человека. Сделаем 4 секции, в каждой из которых возрастной диапазон составит 25 лет.

CREATE TABLE MOCKDATA0 PARTITION OF MOCKDATA

FOR VALUES FROM ('0') TO ('24');

CREATE TABLE MOCKDATA25 PARTITION OF MOCKDATA

FOR VALUES FROM ('25') TO ('49');

CREATE TABLE MOCKDATA50 PARTITION OF MOCKDATA

FOR VALUES FROM ('50') TO ('74');

CREATE TABLE MOCKDATA75 PARTITION OF MOCKDATA

FOR VALUES FROM ('75') TO ('100');

Как говорилось ранее, границы соседних секций могут определяться одинаковыми значениями, так как верхние границы не включаются в диапазон. При добавлении в родительскую таблицу данных, которые не соответствуют ни одной из существующих секций, произойдёт ошибка; подходящую секцию нужно создавать вручную.

1. Создайте в секционируемой таблице индекс по ключевому столбцу, а также любые другие индексы, которые могут понадобиться. (Индекс по ключу, строго говоря, создавать не обязательно, но в большинстве случаев он будет полезен.) При этом автоматически будет создан соответствующий индекс в каждой секции и все секции, которые вы будете создавать или присоединять позднее, тоже будут содержать такой индекс.

CREATE INDEX ON MOCKDATA(age);

## Оценка скорости запросов

Необходимо составить несколько запросов к различным секциям таблицы. Для замера времени запроса будем использовать инструмент \timing on. Результаты выполнения представлены в таблице 1.

Таблица 1 ― Оценка скорости запросов на основе секционирования

|  |  |  |
| --- | --- | --- |
| Запрос | Время выполнения с секционированием (мс) | Время выполнения без секционирования (мс) |
| select \* from mockdata0 where gender = 'M' and country = ‘Russia’; | 0.443 ms | 95.745 ms |
| select \* from mockdata25 where gender = 'F' and country = ‘Albania’; | 0.810 ms | 38.482 ms |
| select mockdata0.country from mockdata25, mockdata0 where mockdata0.country = mockdata25.country; | 171.266 ms | 74.287 ms |

По результатам исследования времени проведения запросов к таблицам можно сделать вывод, что секционирование действительно ускоряет работу с таблицами в рамках одной секции. При получении данных из разных секций производительность не улучшается.

## Индексирование

Далее исследуем поиск с использованием индексов. Индексы — это традиционное средство увеличения производительности БД. Используя индекс, сервер баз данных может находить и извлекать нужные строки гораздо быстрее, чем без него. Однако с индексами связана дополнительная нагрузка на СУБД в целом, поэтому применять их следует обдуманно. По умолчанию в Postgre создается индекс типа Б-дерево. Он является наиболее эффективным и используется чаще всего.

Требуется проверить 5 различных ситуаций:

* Без индекса
* С простым индексом
* С уникальным индексом
* С индексом по выражению
* С индексом по функции

Для этого создадим 5 одинаковых таблиц следующего вида:

create table T1 (

    F1 int,

    F2 int

);

Теперь для каждой таблицы создадим свой индекс, который требуется по заданию. Для первой таблицы нет индекса, для второй таблицы обычный индекс.

drop index T2\_F1\_i;

create index T2\_F1\_i on T2(F1);

Для третьей таблицы требуется уникальный индекс. В настоящее время уникальными могут быть только индексы-B-деревья. Если индекс создаётся как уникальный, в таблицу нельзя будет добавить несколько строк с одинаковыми значениями ключа индекса. При этом значения NULL считаются не равными друг другу. Когда для таблицы определяется ограничение уникальности или первичный ключ, PostgreSQL автоматически создаёт уникальный индекс по всем столбцам, составляющим это ограничение или первичный ключ (индекс может быть составным). Такой индекс и является механизмом, который обеспечивает выполнение ограничения. Уникальность индекса достигается за счет назначения полю признака первичного ключа.

drop table if exists T3;

create table T3 (

    F1 BIGSERIAL NOT NULL PRIMARY KEY,

    F2 int

);

drop index T3\_F1\_i;

create unique index T3\_F1\_i on T3(F1);

Для четвертой таблицы требуется создать индекс по выражению. Индекс можно создать не только по столбцу нижележащей таблицы, но и по функции или скалярному выражению с одним или несколькими столбцами таблицы. Это позволяет быстро находить данные в таблице по результатам вычислений. Например, для сравнений без учёта регистра символов часто используется функция lower. В данном случае число, внесенное в таблицу, будет представляться, как текст.

drop index T4\_F1\_i;

create index T4\_F1\_i on T4(lower(F1::text));

Для пятой таблицы требуется создать индекс по функции. Создадим функцию, которая будет проверять число на четность и возвращать либо его само, либо ближайшее у нему четное. Для корректной работы такого индекса функция должна обладать свойством IMMUTABLE.

create or replace function f5 (int) returns int as

$$

    begin

        if mod($1, 2) = 0

        then return $1;

        else return $1-1;

        end if;

    end;

$$ language plpgsql immutable;

drop index T5\_F1\_i;

create index T5\_F1\_i on T5(f5(F1));

## Оценка скорости запроса SELECT

Было создано 5 таблиц для хранения результатов тестирования. В приложении 1 представлен скрипт, который выполняет запросы ко всем 5 таблицам. Количество записей в таблицах меняется от 10 до 100000. Всего делается по 50 запросов каждый раз и затем считается среднее время. В таблице 2 приведены результаты эксперимента.

Таблица 2 ― Оценка скорости запросов на основе индекса

| Количество записей в таблице | Время выполнения (мс) | | | | |
| --- | --- | --- | --- | --- | --- |
| без индекса | с обычным индексом | с уникальным индексом | с индексом по выражению | с индексом по функции |
| 10 | 5300 | 5014 | 6908 | 5081 | 4998 |
| 50 | 5351 | 5018 | 5171 | 4968 | 5044 |
| 100 | 5069 | 5082 | 5129 | 5046 | 5298 |
| 500 | 5212 | 5248 | 5567 | 5316 | 5147 |
| 1000 | 5540 | 5538 | 5790 | 5447 | 5524 |
| 5000 | 7466 | 7088 | 7561 | 7254 | 7656 |
| 10000 | 9584 | 9881 | 9631 | 10169 | 10099 |
| 50000 | 26694 | 27219 | 28721 | 27900 | 26801 |
| 100000 | 55581 | 51476 | 51024 | 50426 | 51595 |
| 500000 | 246682 | 222331 | 242297 | 225994 | 224845 |
| 1000000 | 660758 | 513836 | 556743 | 586163 | 607328 |

В соответствии с таблицей был построен график зависимости скорости запроса от количества записей при различных индексах (рисунок 5). Результаты примерно одинаковые, возможно из-за малого количества данных.

Рисунок 5 ― График зависимости

## Оценка скорости запроса INSERT

Алгоритм аналогичен оному в предыдущем пункте: таблица заполняется необходимым количеством записей, после чего совершается 50 запросов к ней. В таблице 3 приведены результаты эксперимента.

Таблица 3 ― Оценка скорости запросов на основе индекса

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Количество записей в таблице | Время выполнения (мс) | | | | |
| без индекса | с обычным индексом | с уникальным индексом | с индексом по выражению | с индексом по функции |
| 10 | 25 | 52 | 21 | 47 | 45 |
| 50 | 7 | 31 | 14 | 32 | 50 |
| 100 | 16 | 57 | 26 | 48 | 42 |
| 500 | 6 | 39 | 1384 | 75 | 49 |
| 1000 | 12 | 35 | 22 | 31 | 36 |
| 5000 | 13 | 25 | 36 | 23 | 23 |
| 10000 | 10 | 37 | 22 | 17 | 17 |
| 50000 | 7 | 13 | 14 | 14 | 13 |
| 100000 | 7 | 16 | 18 | 15 | 13 |
| 500000 | 12 | 18 | 14 | 14 | 15 |
| 1000000 | 10 | 17 | 17 | 17 | 13 |

В соответствии с таблицей был построен график зависимости скорости запроса от количества записей при различных индексах (рисунок 6). Операция вставки в таблицу проходит достаточно болезненно, поскольку тогда необходимо перестраивать индексы. Особенно это видно по вставке при 100 уже имеющихся элементах в таблице. Судя по всему, 100 является некоторым граничным значением, при котором нужно перестроить действительно много индексов. В то время, как при 10 и 50 записях перестройка небольшая, а при количестве записей более или равно 500 индексы уже подобраны оптимально, так что перестройка минимальна или вообще отсутствует. Соответственно, для варианта без индекса вставка самая быстрая, для остальных таблиц время в 1,5-2 раза медленнее, причем между собой все индексированные таблицы дают примерно одни и те же результаты.

Рисунок 6 — График зависимости

## Оценка скорости запроса UPDATE

Алгоритм аналогичен оному в предыдущем пункте: таблица заполняется необходимым количеством записей, после чего совершается 50 запросов к ней. В таблице 4 приведены результаты эксперимента.

Таблица 4 ― Оценка скорости запросов на основе индекса

| Количество записей в таблице | Время выполнения (мс) | | | | |
| --- | --- | --- | --- | --- | --- |
| без индекса | с обычным индексом | с уникальным индексом | с индексом по выражению | с индексом по функции |
| 10 | 211480 | 211621 | 235983 | 161480 | 137723 |
| 50 | 208284 | 207058 | 237678 | 159131 | 137233 |
| 100 | 205136 | 204957 | 239261 | 161207 | 137952 |
| 500 | 228027 | 216736 | 238102 | 170485 | 138066 |
| 1000 | 201737 | 213550 | 234005 | 178997 | 144139 |
| 5000 | 231030 | 211400 | 242146 | 208073 | 167849 |
| 10000 | 259883 | 207933 | 247356 | 214241 | 199415 |
| 50000 | 408879 | 237515 | 257186 | 224749 | 221765 |
| 100000 | 807908 | 247702 | 281532 | 257144 | 259879 |
| 500000 | 408879 | 408009 | 445209 | 440919 | 417532 |
| 1000000 | 807908 | 797447 | 721281 | 725414 | 709437 |

В соответствии с таблицей был построен график зависимости скорости запроса от количества записей при различных индексах (рисунок 7). Операция обновления аналогично операции вставки требует изменения индексов. На первых 1000 записях разница слабо выражена, далее график начинает возрастать. В некоторых ситуациях поиск с индексом может оказаться выгоднее, однако разница не сильно выражена, что может быть связано с малым количеством данных.

Рисунок 7 — График зависимости

# Вывод

В ходе выполнения данной лабораторной работы были получены навыки оценки размера физического хранилища данных и работы с ним, изучено, а также реализовано и оценено в контексте селективных запросов секционирование таблицы. Также были изучены некоторые типы индексов в PostgreSQL. Были получены практические навыки реализации и работы с индексами, оценена польза индексов в производительности запросов выборки и их негативное влияние на производительность операций вставки и обновления таблицы.

ПРИЛОЖЕНИЕ 0

drop table if exists T1;

create table T1 (

    F1 int,

    F2 int

);

drop table if exists T2;

create table T2 (

    F1 int,

    F2 int

);

drop table if exists T3;

create table T3 (

    F1 BIGSERIAL NOT NULL PRIMARY KEY,

    F2 int

);

drop table if exists T4;

create table T4 (

    F1 int,

    F2 int

);

drop table if exists T5;

create table T5 (

    F1 int,

    F2 int

);

drop index T2\_F1\_i;

create index T2\_F1\_i on T2(F1);

drop index T3\_F1\_i;

create unique index T3\_F1\_i on T3(F1);

drop index T4\_F1\_i;

create index T4\_F1\_i on T4(lower(F1::text));

drop index T5\_F1\_i;

create index T5\_F1\_i on T5(f5(F1));

drop table if exists results\_T1;

create table results\_T1 (

    num\_of\_rows int,

    time int

);

drop table if exists results\_T2;

create table results\_T2 (

    num\_of\_rows int,

    time int

);

drop table if exists results\_T3;

create table results\_T3 (

    num\_of\_rows int,

    time int

);

drop table if exists results\_T4;

create table results\_T4 (

    num\_of\_rows int,

    time int

);

drop table if exists results\_T5;

create table results\_T5 (

    num\_of\_rows int,

    time int

);

# ПРИЛОЖЕНИЕ 1

create or replace PROCEDURE test\_perfomance(rows\_cnt integer, opsCount integer)

language plpgsql as

$$

DECLARE

t int;

i int;

randvalF1 int;

randvalF2 int;

startTime TIMESTAMP;

endTime TIMESTAMP;

r\_value int;

BEGIN

WHILE rows\_cnt <= 1000000 LOOP

DELETE FROM T1;

DELETE FROM T2;

DELETE FROM T3;

DELETE FROM T4;

DELETE FROM T5;

i := 0;

WHILE i < rows\_cnt LOOP

randvalF1 := round(random() \* 300000);

randvalF2 := round(random() \* 300000);

INSERT INTO T1(F1, F2) VALUES (randvalF1, randvalF2);

INSERT INTO T2(F1, F2) VALUES (randvalF1, randvalF2);

INSERT INTO T3(F2) VALUES (randvalF2);

INSERT INTO T4(F1, F2) VALUES (randvalF1, randvalF2);

INSERT INTO T5(F1, F2) VALUES (randvalF1, randvalF2);

i := i + 1;

END LOOP;

--FOR T1

i := 0;

startTime := clock\_timestamp();

WHILE i < opsCount LOOP

PERFORM F1 FROM T1 WHERE F1 = round(random() \* 300000);

i := i + 1;

END LOOP;

endTime := clock\_timestamp();

t := (1000000 \* (EXTRACT(EPOCH FROM endTime) - EXTRACT(EPOCH FROM startTime))) / opsCount;

INSERT INTO results\_T1(num\_of\_rows, time) VALUES (rows\_cnt,t);

--FOR T2

i := 0;

startTime := clock\_timestamp();

WHILE i < opsCount LOOP

PERFORM F1 FROM T1 WHERE F1 = round(random() \* 300000);

i := i + 1;

END LOOP;

endTime := clock\_timestamp();

t := (1000000 \* (EXTRACT(EPOCH FROM endTime) - EXTRACT(EPOCH FROM startTime))) / opsCount;

INSERT INTO results\_T2(num\_of\_rows, time) VALUES (rows\_cnt,t);

--FOR T3

i := 0;

startTime := clock\_timestamp();

WHILE i < opsCount LOOP

PERFORM F1 FROM T3 WHERE F1 = round(random() \* 300000);

i := i + 1;

END LOOP;

endTime := clock\_timestamp();

t := (1000000 \* (EXTRACT(EPOCH FROM endTime) - EXTRACT(EPOCH FROM startTime))) / opsCount;

INSERT INTO results\_T3(num\_of\_rows, time) VALUES (rows\_cnt,t);

--FOR T4

i := 0;

startTime := clock\_timestamp();

WHILE i < opsCount LOOP

PERFORM F1 FROM T4 WHERE F1 = round(random() \* 300000);

i := i + 1;

END LOOP;

endTime := clock\_timestamp();

t := (1000000 \* (EXTRACT(EPOCH FROM endTime) - EXTRACT(EPOCH FROM startTime))) / opsCount;

INSERT INTO results\_T4(num\_of\_rows, time) VALUES (rows\_cnt,t);

--FOR T5

i := 0;

startTime := clock\_timestamp();

WHILE i < opsCount LOOP

PERFORM F1 FROM T5 WHERE F1 = round(random() \* 300000);

i := i + 1;

END LOOP;

endTime := clock\_timestamp();

t := (1000000 \* (EXTRACT(EPOCH FROM endTime) - EXTRACT(EPOCH FROM startTime))) / opsCount;

INSERT INTO results\_T5(num\_of\_rows, time) VALUES (rows\_cnt,t);

IF rows\_cnt = 10 OR rows\_cnt = 100 OR rows\_cnt = 1000 OR rows\_cnt = 10000 OR rows\_cnt = 100000 THEN

rows\_cnt := rows\_cnt \* 5;

ELSE

rows\_cnt := rows\_cnt \* 2;

END IF;

END LOOP;

END

$$;

CALL test\_perfomance(10, 50);

SELECT \* FROM results\_T1;

SELECT \* FROM results\_T2;

SELECT \* FROM results\_T3;

SELECT \* FROM results\_T4;

SELECT \* FROM results\_T5;

# ПРИЛОЖЕНИЕ 2

create or replace PROCEDURE test\_insert(rows\_cnt integer, opsCount integer)

language plpgsql as

$$

DECLARE

t int;

i int;

randvalF1 int;

randvalF2 int;

startTime TIMESTAMP;

endTime TIMESTAMP;

r\_value int;

BEGIN

DELETE FROM T1;

DELETE FROM T2;

DELETE FROM T3;

DELETE FROM T4;

DELETE FROM T5;

WHILE rows\_cnt <= 1000000 LOOP

i := 0;

WHILE i < rows\_cnt LOOP

randvalF1 := round(random() \* 300000);

randvalF2 := round(random() \* 300000);

INSERT INTO T1(F1, F2) VALUES (randvalF1, randvalF2);

INSERT INTO T2(F1, F2) VALUES (randvalF1, randvalF2);

INSERT INTO T3(F2) VALUES (randvalF2);

INSERT INTO T4(F1, F2) VALUES (randvalF1, randvalF2);

INSERT INTO T5(F1, F2) VALUES (randvalF1, randvalF2);

i := i + 1;

END LOOP;

--FOR T1

i := 0;

startTime := clock\_timestamp();

WHILE i < opsCount LOOP

randvalF1 := round(random() \* 300000);

randvalF2 := round(random() \* 300000);

INSERT INTO T1(F1, F2) VALUES (randvalF1, randvalF2);

i := i + 1;

END LOOP;

endTime := clock\_timestamp();

t := (1000000 \* (EXTRACT(EPOCH FROM endTime) - EXTRACT(EPOCH FROM startTime))) / opsCount;

INSERT INTO results\_T1(num\_of\_rows, time) VALUES (rows\_cnt,t);

--FOR T2

i := 0;

startTime := clock\_timestamp();

WHILE i < opsCount LOOP

randvalF1 := round(random() \* 300000);

randvalF2 := round(random() \* 300000);

INSERT INTO T2(F1, F2) VALUES (randvalF1, randvalF2);

i := i + 1;

END LOOP;

endTime := clock\_timestamp();

t := (1000000 \* (EXTRACT(EPOCH FROM endTime) - EXTRACT(EPOCH FROM startTime))) / opsCount;

INSERT INTO results\_T2(num\_of\_rows, time) VALUES (rows\_cnt,t);

--FOR T3

i := 0;

startTime := clock\_timestamp();

WHILE i < opsCount LOOP

randvalF1 := round(random() \* 300000);

randvalF2 := round(random() \* 300000);

INSERT INTO T3(F2) VALUES (randvalF2);

i := i + 1;

END LOOP;

endTime := clock\_timestamp();

t := (1000000 \* (EXTRACT(EPOCH FROM endTime) - EXTRACT(EPOCH FROM startTime))) / opsCount;

INSERT INTO results\_T3(num\_of\_rows, time) VALUES (rows\_cnt,t);

--FOR T4

i := 0;

startTime := clock\_timestamp();

WHILE i < opsCount LOOP

randvalF1 := round(random() \* 300000);

randvalF2 := round(random() \* 300000);

INSERT INTO T2(F1, F2) VALUES (randvalF1, randvalF2);

i := i + 1;

END LOOP;

endTime := clock\_timestamp();

t := (1000000 \* (EXTRACT(EPOCH FROM endTime) - EXTRACT(EPOCH FROM startTime))) / opsCount;

INSERT INTO results\_T4(num\_of\_rows, time) VALUES (rows\_cnt,t);

--FOR T5

i := 0;

startTime := clock\_timestamp();

WHILE i < opsCount LOOP

randvalF1 := round(random() \* 300000);

randvalF2 := round(random() \* 300000);

INSERT INTO T2(F1, F2) VALUES (randvalF1, randvalF2);

i := i + 1;

END LOOP;

endTime := clock\_timestamp();

t := (1000000 \* (EXTRACT(EPOCH FROM endTime) - EXTRACT(EPOCH FROM startTime))) / opsCount;

INSERT INTO results\_T5(num\_of\_rows, time) VALUES (rows\_cnt,t);

IF rows\_cnt = 10 OR rows\_cnt = 100 OR rows\_cnt = 1000 OR rows\_cnt = 10000 OR rows\_cnt = 100000 THEN

rows\_cnt := rows\_cnt \* 5;

ELSE

rows\_cnt := rows\_cnt \* 2;

END IF;

END LOOP;

END

$$;

CALL test\_insert(10, 50);

SELECT \* FROM results\_T1;

SELECT \* FROM results\_T2;

SELECT \* FROM results\_T3;

SELECT \* FROM results\_T4;

SELECT \* FROM results\_T5;

# ПРИЛОЖЕНИЕ 3

create or replace PROCEDURE test\_update(rows\_cnt integer, opsCount integer)

language plpgsql as

$$

DECLARE

t int;

i int;

randvalF1 int;

randvalF2 int;

startTime TIMESTAMP;

endTime TIMESTAMP;

r\_value int;

BEGIN

WHILE rows\_cnt <= 1000000 LOOP

DELETE FROM T1;

DELETE FROM T2;

DELETE FROM T3;

DELETE FROM T4;

DELETE FROM T5;

i := 0;

WHILE i < rows\_cnt LOOP

randvalF1 := round(random() \* 300000);

randvalF2 := round(random() \* 300000);

INSERT INTO T1(F1, F2) VALUES (randvalF1, randvalF2);

INSERT INTO T2(F1, F2) VALUES (randvalF1, randvalF2);

INSERT INTO T3(F2) VALUES (randvalF2);

INSERT INTO T4(F1, F2) VALUES (randvalF1, randvalF2);

INSERT INTO T5(F1, F2) VALUES (randvalF1, randvalF2);

i := i + 1;

END LOOP;

--FOR T1

i := 0;

startTime := clock\_timestamp();

WHILE i < opsCount LOOP

UPDATE T1 SET F1 = round(random() \* 300000) WHERE F1 = round(random() \* 300000);

i := i + 1;

END LOOP;

endTime := clock\_timestamp();

t := (1000000 \* (EXTRACT(EPOCH FROM endTime) - EXTRACT(EPOCH FROM startTime))) / opsCount;

INSERT INTO results\_T1(num\_of\_rows, time) VALUES (rows\_cnt,t);

--FOR T2

i := 0;

startTime := clock\_timestamp();

WHILE i < opsCount LOOP

UPDATE T2 SET F1 = round(random() \* 300000) WHERE F1 = round(random() \* 300000);

i := i + 1;

END LOOP;

endTime := clock\_timestamp();

t := (1000000 \* (EXTRACT(EPOCH FROM endTime) - EXTRACT(EPOCH FROM startTime))) / opsCount;

INSERT INTO results\_T2(num\_of\_rows, time) VALUES (rows\_cnt,t);

--FOR T3

i := 0;

startTime := clock\_timestamp();

WHILE i < opsCount LOOP

UPDATE T3 SET F2 = round(random() \* 300000) WHERE F1 = round(random() \* 300000);

i := i + 1;

END LOOP;

endTime := clock\_timestamp();

t := (1000000 \* (EXTRACT(EPOCH FROM endTime) - EXTRACT(EPOCH FROM startTime))) / opsCount;

INSERT INTO results\_T3(num\_of\_rows, time) VALUES (rows\_cnt,t);

--FOR T4

i := 0;

startTime := clock\_timestamp();

WHILE i < opsCount LOOP

UPDATE T4 SET F1 = round(random() \* 300000) WHERE F1 = round(random() \* 300000);

i := i + 1;

END LOOP;

endTime := clock\_timestamp();

t := (1000000 \* (EXTRACT(EPOCH FROM endTime) - EXTRACT(EPOCH FROM startTime))) / opsCount;

INSERT INTO results\_T4(num\_of\_rows, time) VALUES (rows\_cnt,t);

--FOR T5

i := 0;

startTime := clock\_timestamp();

WHILE i < opsCount LOOP

UPDATE T5 SET F1 = round(random() \* 300000) WHERE F1 = round(random() \* 300000);

i := i + 1;

END LOOP;

endTime := clock\_timestamp();

t := (1000000 \* (EXTRACT(EPOCH FROM endTime) - EXTRACT(EPOCH FROM startTime))) / opsCount;

INSERT INTO results\_T5(num\_of\_rows, time) VALUES (rows\_cnt,t);

IF rows\_cnt = 10 OR rows\_cnt = 100 OR rows\_cnt = 1000 OR rows\_cnt = 10000 OR rows\_cnt = 100000 THEN

rows\_cnt := rows\_cnt \* 5;

ELSE

rows\_cnt := rows\_cnt \* 2;

END IF;

END LOOP;

END

$$;

CALL test\_update(10, 50);

SELECT \* FROM results\_T1;

SELECT \* FROM results\_T2;

SELECT \* FROM results\_T3;

SELECT \* FROM results\_T4;

SELECT \* FROM results\_T5;