**Task 7 (VACUUM)**

Папка задания в репозитории:

<https://github.com/artemgur/AdvancedPostgreSQL/tree/master/Task%207%20(Vacuum)>

Flyway-скрипты хранятся в следующей папке:

<https://github.com/artemgur/AdvancedPostgreSQL/tree/master/flyway/sql>

Flyway-скрипты, относящиеся к этому заданию, имеют номер версии вида 7.\*.

**Создание таблицы, вставка 1 строки**

Создание таблицы с отключенным AUTOVACUUM:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/flyway/sql/V7.0__CreateTable.sql>

Таблицу я сделал unlogged. Это значительно ускорит подготовительные операции, но не повлияет на результаты задания, так как наша задача — исследовать размер таблицы и время выполнения SELECT.

Вставка строки в таблицу:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/flyway/sql/V7.1__InsertRow.sql>

**Размер таблицы после вставки 1 строки**

Здесь и далее для измерения размера таблицы использовался следующий скрипт (измененный скрипт с лекции 1):

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/Task%207%20(Vacuum)/TableSize.sql>

Количество страниц таблицы я определял как размер таблицы в байтах / 8192, так как [согласно документации](https://postgrespro.ru/docs/postgresql/14/planner-stats), relpages иногда содержит устаревшие значения.

* 8 кБ
* 1 страница

**Скорость выполнения SELECT после вставки 1 строки**

Здесь и далее с помощью команды:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/Task%207%20(Vacuum)/PgBenchSelect.sh>

запускался скрипт:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/Task%207%20(Vacuum)/PgBenchSelect.sql>

* 0.179 ms — latency average
* 5592.343761 — tps (including connections establishing)
* 5605.946895 — tps (excluding connections establishing)

**UPDATE строки**

Для обновления строки 1000000 раз использовался PgBench. С помощью команды:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/Task%207%20(Vacuum)/PgBenchUpdate.sh>

запускался скрипт:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/Task%207%20(Vacuum)/PgBenchUpdate.sql>

10 клиентов вставили по 100 000 строк.

**Размер таблицы после обновления строки**

* 6064 кБ
* 758 страниц

**Скорость выполнения SELECT после обновления строки**

* 0.202 ms — latency average
* 4943.020889 — tps (including connections establishing)
* 4953.926512 — tps (excluding connections establishing)

Таким образом, после многократного UPDATE строки значительно вырос объем таблицы, и запросы SELECT стали выполняться медленнее.

**VACUUM**

VACUUM проводился с помощью следующей команды:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/Task%207%20(Vacuum)/Vacuum.sql>

**Размер таблицы после VACUUM**

* 6064 кБ
* 758 страниц

**Скорость выполнения SELECT после VACUUM**

* 0.184 ms — latency average
* 5432.430650 — tps (including connections establishing)
* 5445.385559 — tps (excluding connections establishing)

VACUUM не уменьшил размер таблицы, но сократил время выполнения запросов SELECT.

**VACUUM FULL**

VACUUM FULL проводился с помощью следующей команды:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/Task%207%20(Vacuum)/VacuumFull.sql>

**Размер таблицы после VACUUM FULL**

* 8 кБ
* 1 страница

**Скорость выполнения SELECT после VACUUM FULL**

* 0.172 ms — latency average
* 5809.287543 — tps (including connections establishing)
* 5825.691254 — tps (excluding connections establishing)

После VACUUM FULL объем таблицы стал таким же, как до многократного UPDATE. Скорость выполнения запросов SELECT тоже вернулась на уровень, который был до многократного UPDATE. Таким образом, VACUUM FULL действительно полностью очищает таблицу.