**Task 2 (Unlogged Table)**

Папка задания в репозитории:

<https://github.com/artemgur/AdvancedPostgreSQL/tree/master/PostgreSQL%20Task%202%20(Unlogged)>

**Создание таблиц с помощью Flyway**

Файл миграции:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/flyway/sql/V1.1__createTables.sql>

Запустить миграцию можно, запустив команду flyway migrate в папке flyway репозитория (например, используя файл migrate.sh), так как команда flyway ищет папку sql, содержащую файлы миграций, в рабочей папке.

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

Таблицы содержат следующие столбцы:

* id — bigserial PRIMARY KEY. bigserial имеет то же максимальное значение, что bigint
* time — timestamp
* string — char(10)

Если база данных, на которой мы планируем использовать flyway, не пустая, для инициализации flyway следует выполнить команду flyway baseline (например, используя файл baseline.sh).

**INSERT**

В случае таблицы logged с помощью команды:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/PostgreSQL%20Task%202%20(Unlogged)/PgBenchInsert/PgBenchInsertLogged.sh>

измеряется производительность скрипта:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/PostgreSQL%20Task%202%20(Unlogged)/PgBenchInsert/InsertLogged.sql>

В случае таблицы unlogged с помощью команды:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/PostgreSQL%20Task%202%20(Unlogged)/PgBenchInsert/PgBenchInsertUnlogged.sh>

измеряется производительность скрипта:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/PostgreSQL%20Task%202%20(Unlogged)/PgBenchInsert/InsertUnlogged.sql>

Здесь и далее в pgbench был использован флаг -n (--no-vacuum), так как, согласно документации, он обязателен, если скрипт не затрагивает стандартные таблицы pgbench.

Были получены следующие результаты:

logged:

* 47965 транзакций (было вставлено 47965 строк)
* 6.255 ms — latency average
* 159.880141 — tps (including connections establishing)
* 159.881780 — tps (excluding connections establishing)

unlogged:

* 747163 транзакции (было вставлено 747163 строки)
* 0.402 ms — latency average
* 2490.526039 — tps (including connections establishing)
* 2490.550443 — tps (excluding connections establishing)

Коэффициент для latency:

6.255 / 0.402 ≈ 15.56. unlogged — в 15.56 раз (на 1456%) быстрее.

Коэффициент для tps (excluding connections establishing):

2490.550443 / 159.881780 ≈ 15.58. tps в случае unlogged — в 15.58 раз (на 1458%) быстрее.

**UPDATE**

В случае таблицы logged с помощью команды:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/PostgreSQL%20Task%202%20(Unlogged)/PgBenchUpdate/PgBenchUpdateLogged.sh>

измеряется производительность скрипта:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/PostgreSQL%20Task%202%20(Unlogged)/PgBenchUpdate/UpdateLogged.sql>

В случае таблицы unlogged с помощью команды:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/PostgreSQL%20Task%202%20(Unlogged)/PgBenchUpdate/PgBenchUpdateUnlogged.sh>

измеряется производительность скрипта:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/PostgreSQL%20Task%202%20(Unlogged)/PgBenchUpdate/UpdateUnlogged.sql>

Перед запуском бенчмарка таблицы были заполнены равным количеством строк (10000). Это нужно, чтобы высоты B-tree индексов первичных ключей были равны.

В качестве “переменной-счетчика” использовалась последовательность pgbench\_sequence. На каждой итерации бенчмарка обновлялась строка с id=nextval('pgbench\_sequence').

UPDATE проводился 10000 раз. Каждый раз обновлялась новая строка.

Были получены следующие результаты:

logged:

* 4.788 ms — latency average
* 208.875239 — tps (including connections establishing)
* 208.887916 — tps (excluding connections establishing)

unlogged:

* 0.412 ms — latency average
* 2427.102722 — tps (including connections establishing)
* 2428.719116 — tps (excluding connections establishing)

Коэффициент для latency:

4.788 / 0.412 ≈ 11.62. unlogged — в 11.62 раз (на 1062%) быстрее.

Коэффициент для tps (excluding connections establishing):

2428.719116 / 208.887916 ≈ 11.63. tps в случае unlogged — в 11.63 раз (на 1063%) быстрее.

**DELETE**

В случае таблицы logged с помощью команды:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/PostgreSQL%20Task%202%20(Unlogged)/PgBenchDelete/PgBenchDeleteLogged.sh>

измеряется производительность скрипта:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/PostgreSQL%20Task%202%20(Unlogged)/PgBenchDelete/DeleteLogged.sql>

В случае таблицы unlogged с помощью команды:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/PostgreSQL%20Task%202%20(Unlogged)/PgBenchDelete/PgBenchDeleteUnlogged.sh>

измеряется производительность скрипта:

<https://github.com/artemgur/AdvancedPostgreSQL/blob/master/PostgreSQL%20Task%202%20(Unlogged)/PgBenchDelete/DeleteUnlogged.sql>

Как и в случае UPDATE, перед запуском бенчмарка таблицы были заполнены равным количеством строк (10000).

Как и в случае UPDATE, в качестве “переменной-счетчика” использовалась последовательность pgbench\_sequence.

Также я пробовал вариант, когда на каждой итерации pgbench с помощью запроса вида SELECT id FROM table\_name LIMIT 1 берется первый попавшийся id строки. Но оказалось, что метод последовательности работает быстрее. Более того, так как UPDATE и DELETE используют один и тот же метод отбора строк, их можно сравнивать.

DELETE проводился 10000 раз. Каждый раз удалялась новая строка.

Были получены следующие результаты:

logged:

* 4.721 ms — latency average
* 211.818278 — tps (including connections establishing)
* 211.837005 — tps (excluding connections establishing)

unlogged:

* 0.421 ms — latency average
* 2376.346545 — tps (including connections establishing)
* 2377.886730 — tps (excluding connections establishing)

Коэффициент для latency:

4.721 / 0.421 ≈ 11.21. unlogged — в 11.21 раз (на 1021%) быстрее.

Коэффициент для tps (excluding connections establishing):

2377.886730 / 211.837005 ≈ 11.23. tps в случае unlogged — в 11.23 раз (на 1023%) быстрее.

**Выводы и интересные моменты**

Очевидно, что как минимум для одиночных запросов практически все время занимает логирование. Таким образом, unlogged-таблицы позволяют ускорить многие транзакции в несколько раз и являются очень полезным инструментом в случаях, когда восстановление данных не важно.

Интересно, что для таблицы logged UPDATE работает быстрее, чем INSERT, даже несмотря на overhead от использования последовательности, хотя UPDATE — примерно то же, что INSERT+DELETE.

Возможно, это связано с особенностями логирования.

В то же время, INSERT, UPDATE и DELETE в случае таблицы unlogged работают за практически одинаковое время, несмотря на то что UPDATE — примерно то же, что INSERT+DELETE, а DELETE — гораздо более простая операция, чем INSERT и UPDATE.

Таким образом, похоже, для одиночных запросов большую часть времени занимает не сама операция, а технические моменты, такие как открытие и закрытие файлов.