**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение**

**Высшего профессионального образования**

**Московский технический университет связи и информатики**

Факультет повышения квалификации

**Лабораторная работа №2**

**по дисциплине**

**«Интеллектуальные системы»**

Выполнил:

магистрант гр. 3мпп1901

Каргальцев М. М.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись дата

Проверил:

к.т.н., доцент каф. МКиИТ

Махров С.С.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись дата

**Москва 2020**

# Цель работы:

Целью работы является изучение возможностей профилирования и оптимизации запросов в реляционных базах данных.

# Задание:

1. Скачать выгрузку БД об авиаперелетах за <https://postgrespro.ru/education/demodb> . Необходима база за год demo-big.zip
2. Изучить цикл статей про оптимизацию и профилирование запросов:

* <https://habr.com/ru/post/203320/>
* <https://habr.com/en/post/203386/>
* <https://habr.com/en/post/203484/>

1. Придумать 1 запрос, в котором участвует несколько таблиц, и чтобы его выполнение длилось не менее 1 секунды.
2. Для запроса показать результат EXPLAIN, а затем EXPLAIN (ANALYZE).
3. Создайте индексы на поля, которые участвуют запросе и примените последовательно следующие стратегии оптимизации:

* Index Scan
* Bitmap Index Scan
* Index Only Scan

1. Исследовать, как меняется время исполнения запроса, если его исполнять несколько раз подряд? Как используется кэш? Делать запрос с указанием опции BUFFERS.
2. Предоставить отчет в direct slack <https://join.slack.com/t/intsystems/shared_invite/zt-dgvb13xu-VJ1itP7HN33PoJXcMLGb4Q>

# Ход выполнения работы:

(Содержимое отчета:

1. ER-диаграмма;
2. Указать созданный запрос, время исполнения которого не менее 1 секунды;
3. Тестовые запросы CRUD и результат выполнения каждого запроса.
4. Для запроса показать результат EXPLAIN, а затем EXPLAIN (ANALYZE).
5. Описание того, на какие поля создавали индексы и как применяли различные стартегии оптимизации? Какой получался результат?
6. Результаты исследования использования кэша.

# Ключевые вопросы:

1. Что такое EXPLAIN PLAN?
2. Зачем нужны операторы EXPLAIN и ANALYZE? Можно ли их применять отдельно?
3. Зачем нужен BUFFERS?
4. Зачем нужны индексы?
5. Какие типы индексов есть в PostgreSQL?
6. Как работает кэш в PostgreSQL?

# Вывод:

В результате выполнения лабораторной работы, были изучены механизмы планировщика ядра Postgres. Появились практические навыки работы с командой EXPLAIN и ее вариациями, а также созданием и использованием индексов Postgres.

Отчет по работе.

1.

**select** \*

**from** bookings.ticket\_flights tf

**join** bookings.tickets t

**on** t.ticket\_no = tf.ticket\_no

**join** bookings.bookings b

**on** b.book\_ref = t.book\_ref

**where** b.total\_amount < 100.000

2.

**Результат выполнения с** **explain**:

Hash Join (cost=356135.14..815822.26 rows=8391852 width=157)

Hash Cond: (tf.ticket\_no = t.ticket\_no)

-> Seq Scan on ticket\_flights tf (cost=0.00..153851.52 rows=8391852 width=32)

-> Hash (cost=264527.93..264527.93 rows=2949857 width=125)

-> Hash Join (cost=73316.98..264527.93 rows=2949857 width=125)

Hash Cond: (t.book\_ref = b.book\_ref)

-> Seq Scan on tickets t (cost=0.00..78913.57 rows=2949857 width=104)

-> Hash (cost=34558.10..34558.10 rows=2111110 width=21)

-> Seq Scan on bookings b (cost=0.00..34558.10 rows=2111110 width=21)

JIT:

Functions: 19

Options: Inlining true, Optimization true, Expressions true, Deforming true

**Результат выполнения с** **explain**(**analyze**):

Hash Join (cost=356135.14..815822.26 rows=8391852 width=157) (actual time=7251.258..17689.039 rows=8391852 loops=1)

Hash Cond: (tf.ticket\_no = t.ticket\_no)

-> Seq Scan on ticket\_flights tf (cost=0.00..153851.52 rows=8391852 width=32) (actual time=0.025..1134.741 rows=8391852 loops=1)

-> Hash (cost=264527.93..264527.93 rows=2949857 width=125) (actual time=7249.740..7249.741 rows=2949857 loops=1)

Buckets: 32768 Batches: 128 Memory Usage: 3890kB

-> Hash Join (cost=73316.98..264527.93 rows=2949857 width=125) (actual time=1071.347..5382.000 rows=2949857 loops=1)

Hash Cond: (t.book\_ref = b.book\_ref)

-> Seq Scan on tickets t (cost=0.00..78913.57 rows=2949857 width=104) (actual time=0.010..513.975 rows=2949857 loops=1)

-> Hash (cost=34558.10..34558.10 rows=2111110 width=21) (actual time=1070.974..1070.974 rows=2111110 loops=1)

Buckets: 65536 Batches: 64 Memory Usage: 2277kB

-> Seq Scan on bookings b (cost=0.00..34558.10 rows=2111110 width=21) (actual time=0.012..256.094 rows=2111110 loops=1)

Planning Time: 0.693 ms

JIT:

Functions: 19

Options: Inlining true, Optimization true, Expressions true, Deforming true

Timing: Generation 3.561 ms, Inlining 3.741 ms, Optimization 181.398 ms, Emission 111.669 ms, Total 300.370 ms

Execution Time: 18150.654 ms.

3.

**CREATE** **index** IX\_ticket\_no **ON** bookings.ticket\_flights(ticket\_no);

**CREATE** **index** IX\_tickets\_book\_ref **ON** bookings.tickets(book\_ref);

**CREATE** **index** IX\_total\_amount **ON** bookings.bookings(total\_amount);

Для таблицы ticket\_flights был создан индекс IX\_ticket\_no по полю ticket\_no для оптимизации join с таблицей tickets, где поле ticket\_no является первичным ключом. Для таблицы tickets был создан индекс IX\_tickets\_book\_ref по полю book\_ref для оптимизации join с таблицей bookings, где поле book\_ref является первичным ключом.

Индекс IX\_total\_amount был создан для демонстрации оптимизации запросов к таблице bookings.

**explain**

**select** \*

**from** bookings.ticket\_flights tf

**join** bookings.tickets t

**on** t.ticket\_no = tf.ticket\_no

**join** bookings.bookings b

**on** b.book\_ref = t.book\_ref

**where** b.total\_amount < 100.000

**Результат выполнения:**

Nested Loop (cost=1.42..21.74 rows=4 width=157)

-> Nested Loop (cost=0.86..20.93 rows=1 width=125)

-> Index Scan using ix\_total\_amount on bookings b (cost=0.43..8.45 rows=1 width=21)

Index Cond: (total\_amount < 100.000)

-> Index Scan using ix\_tickets\_book\_ref on tickets t (cost=0.43..12.46 rows=2 width=104)

Index Cond: (book\_ref = b.book\_ref)

-> Index Scan using ix\_ticket\_no on ticket\_flights tf (cost=0.56..0.77 rows=4 width=32)

Index Cond: (ticket\_no = t.ticket\_no)

**explain**(**analyse** )

**select** b.total\_amount

**from** bookings.bookings b

**where** b.total\_amount < 90.000

**Результат выполнения:**

Index Only Scan using ix\_total\_amount on bookings b (cost=0.43..8.45 rows=1 width=6) (actual time=0.007..0.007 rows=0 loops=1)

Index Cond: (total\_amount < 90.000)

Heap Fetches: 0

Planning Time: 0.158 ms

Execution Time: 0.026 ms

**explain**(**analyse** )

**select** \*

**from** bookings.bookings b

**where** b.total\_amount < 1000

**Результат выполнения:**

Bitmap Heap Scan on bookings b (cost=4.44..8.45 rows=1 width=21) (actual time=0.008..0.008 rows=0 loops=1)

Recheck Cond: (total\_amount < '1000'::numeric)

-> Bitmap Index Scan on ix\_total\_amount (cost=0.00..4.44 rows=1 width=0) (actual time=0.005..0.005 rows=0 loops=1)

Index Cond: (total\_amount < '1000'::numeric)

Planning Time: 0.135 ms

Execution Time: 0.073 ms

4.

**EXPLAIN** (**ANALYZE**,BUFFERS)

**select** \* **from** bookings.flights f

**Результат выполнения 1 запроса:**

Seq Scan on flights f (cost=0.00..4772.67 rows=214867 width=63) (actual time=0.012..69.546 rows=214867 loops=1)

Buffers: shared hit=37 read=2587

Planning Time: 0.054 ms

Execution Time: 86.888 ms

**Результат выполнения 2 запроса:**

Seq Scan on flights f (cost=0.00..4772.67 rows=214867 width=63) (actual time=0.064..35.855 rows=214867 loops=1)

Buffers: shared hit=2624

Planning Time: 0.114 ms

Execution Time: 55.095 ms

**Результат выполнения 3 запроса:**

Seq Scan on flights f (cost=0.00..4772.67 rows=214867 width=63) (actual time=0.014..37.386 rows=214867 loops=1)

Buffers: shared hit=2624

Planning Time: 0.060 ms

Execution Time: 54.611 ms

Вывод: в первом запросе из кэша Postres было прочитано 37 блоков, о чем говорит запись планировщика: Buffers: shared hit=37. С каждым последующем запросом количество блоков в кэше(оперативной памяти) увеличивается и соответственно планировщик ядра Postgres прочитывает все больше данных из оперативной памяти(кэша), так как это собственно увеличивает производительность. В третьем запросе количество считываемых блоков не изменилось. Это говорит о том, что вся таблица находится в оперативной памяти.

Ответы на вопросы.

1. EXPLAIN PLAN — это план ядра Postgres на выполнение запроса.

2. Команда EXPLAIN выводит «ожидания» планировщика для выполнения запроса. Вариация EXPLAIN(ANALYZE) считывает определённое количество строк таблицы, выбранных случайным образом и таким орбазом собирается статистика значений по каждой из колонок таблицы. Применять отдельно можно, ANALYZE используется для обновления плана запроса таблицы.

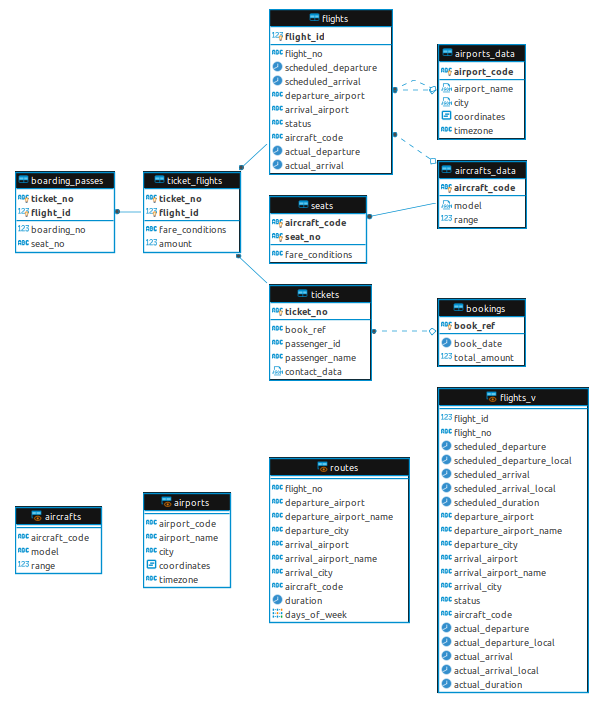
3. Команда BUFFERS позволят получить информацию об использовании кэша Postgres.

4. Индексы - это специальные структуры в базах данных, которые позволяют ускорить поиск и сортировку по определенному полю или набору полей в таблице, а также используются для обеспечения уникальности данных.

5. Типы индексов Postgres: B-дерево, хеш, GiST, SP-GiST, GIN и BRIN. Для разных типов индексов применяются разные алгоритмы, ориентированные на определённые типы запросов. По умолчанию команда CREATE INDEX создаёт индексы типа B-дерево, эффективные в большинстве случаев.

6. Кэш по сути является выделенным местом в оперативной памяти. После каждого запроса ядро Postgres постепенно помещает некоторое количество данных(блоков определенной размерности) в кэш(оперативную память) откуда чтение происходит гораздо быстрее.

ER-диаграмма базы demo.

Схема 1 - ER-диаграмма базы demo