

Министерство науки и высшего образования РФ  
Федеральное государственное автономное  
образовательное учреждение высшего образования  
**«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**  
Институт космических и информационных технологий  
Кафедра «Программная инженерия»

## ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №3

по предмету “Серверное программирование”

Нагрузочное тестирование разработанной архитектуры базы данных

тема

Преподаватель

\_\_\_\_\_

А.А. Даничев

подпись, дата

инициалы, фамилия

Студент ЗКИ21-16БВВ 031625881

05.04.2025

Е.М.Хорошко

подпись, дата

инициалы, фамилия

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
1.   Ход работы.....	4
1.1 Заполнение таблиц базы дополнительными данными Populate() ...	4
1.2 SQL SELECT JOIN запрос для 3х таблиц с условием WHERE.....	7
2.   Добавление индексов на поля, используемые в WHERE.....	8
3.   Замена типа одного из индексов на bitmap.....	8
4.   Добавление индексов на ссылочные поля, используемые в JOIN.....	9
5.   QuaryPlan. Заполнение сравнительной таблицы.....	10
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	12

## ВВЕДЕНИЕ

### Цели и задачи работы:

В рамках данной практической работы необходимо провести «нагрузочное» тестирование разработанной архитектуры БД, а также произвести работы по оптимизации запросов и структуры данных.

В ходе практической работы необходимо выполнить следующие задачи:

1. Используйте Populate(), чтобы добавить не менее 500000 новых объектов таблиц сущностей. Для корректного тестирования таблицы-справочники, в которых потенциально может быть более 10 значений так же необходимо заполнить дополнительными тестовыми данными.
2. Придумайте запрос, который может быть использован в разрабатываемой системе. Запрос должен включать в себя не менее 3х таблиц, а также не менее 3х условий в WHERE для фильтрации данных.
3. Установите режим отображения на Display Mode, запустите запрос и запишите значения производительности и количество глобальных ссылок в таблицу в конце этого упражнения. Нажмите Show Query Plan и прочтите план.
4. Добавьте индекс на свойство используемое в WHERE
5. В Портале просмотрите таблицу, для свойства которой был создан индекс и нажмите Действия Перестроить индексы. Обновите страницу. Если для столбцов нет значений Избирательности, нажмите Настроить таблицу и кнопку Настроить таблицу.
6. Выполните запрос ещё раз. Запишите значения производительности и количество глобальных ссылок. Нажмите Show Query Plan и прочтите план. Обратите внимание на использование индекса на выбранном поле.

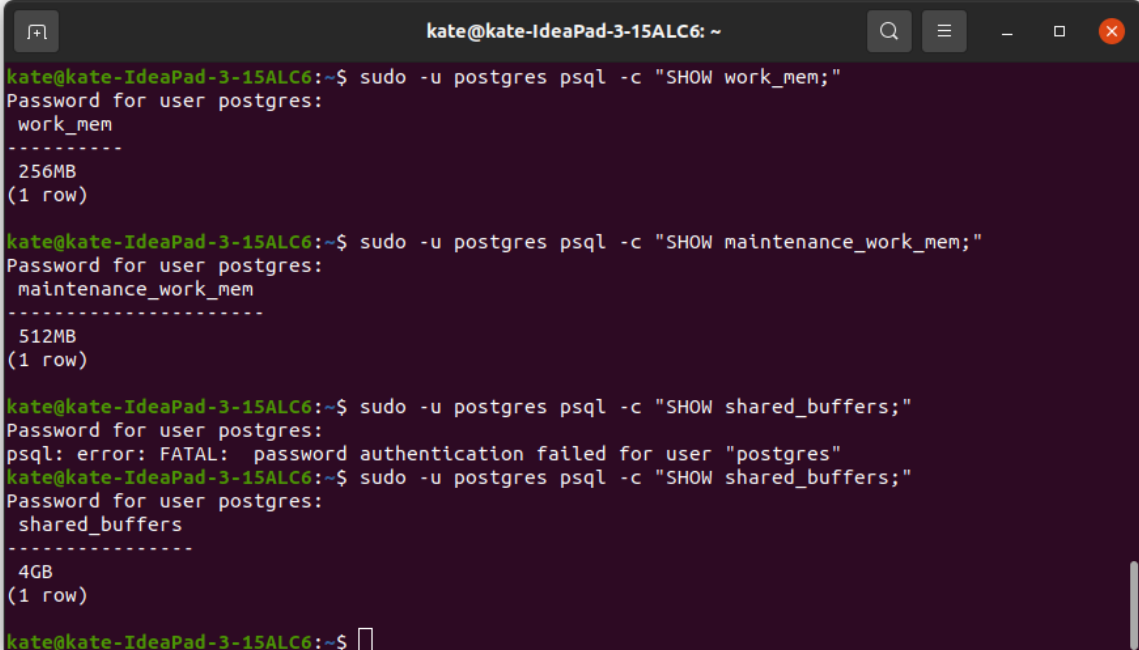
7. Добавьте индексы к оставшимся свойствам, используемым в WHERE и скомпилируйте. Снова нажмите Перестроить индексы.
8. Снова запустите запрос и запишите значения производительности и количество глобальных ссылок. Нажмите Show Query Plan и прочтите план. Обратите внимание на использование всех индексов.
9. Замените Type одного из индексов на bitmap и скомпилируйте.  
Index FIELDNAMEIndex on FIELDNAME [Type = bitmap];
10. Снова нажмите Действия Перестроить индексы.
11. Снова запустите запрос и запишите значения производительности и количество глобальных ссылок. Нажмите Show Query Plan и прочтите план.
12. Добавьте индексы на ссылочные поля, используемые для объединения в JOIN. Повторите перестройку индексов и замер.

## 1. Ход работы

### 1.1 Заполнение таблиц базы дополнительными данными Populate()

В базу данных был добавлен дополнительный словарь Country (Страна производителя). Согласно рекомендациям [2], при наполнении базы большим объемом данных следует выполнить следующие действия по оптимизации для PostgreSQL:

1. Отключить автофиксацию транзакций: загрузка данных должна выполняться в рамках одной транзакции с возможностью отката всей операции в случае ошибки.
2. Использовать COPY вместо INSERT: COPY — это одна команда, оптимизированная для загрузки большого количества строк. Применяя её, нет необходимости отключать автофиксацию транзакций.
3. Удалить индексы.
4. Удалить ограничения внешних ключей.
5. Увеличить maintenance\_work\_mem, max\_wal\_size, shared\_buffers.
6. Назначить параметру wal\_level значение minimal, archive\_mode — off, а max\_wal\_senders — 0. Перезагрузить сервер БД.

A screenshot of a terminal window titled 'kate@kate-IdeaPad-3-15ALC6: ~'. The terminal shows three commands being executed to check PostgreSQL configuration parameters. The first command is 'sudo -u postgres psql -c "SHOW work\_mem;"', which returns 'work\_mem' as '256MB'. The second command is 'sudo -u postgres psql -c "SHOW maintenance\_work\_mem;"', which returns 'maintenance\_work\_mem' as '512MB'. The third command is 'sudo -u postgres psql -c "SHOW shared\_buffers;"', which initially fails with a 'FATAL: password authentication failed for user "postgres"' error, but after re-entering the password, it returns 'shared\_buffers' as '4GB'.

```
kate@kate-IdeaPad-3-15ALC6:~$ sudo -u postgres psql -c "SHOW work_mem;"
Password for user postgres:
work_mem
-----
256MB
(1 row)

kate@kate-IdeaPad-3-15ALC6:~$ sudo -u postgres psql -c "SHOW maintenance_work_mem;"
Password for user postgres:
maintenance_work_mem
-----
512MB
(1 row)

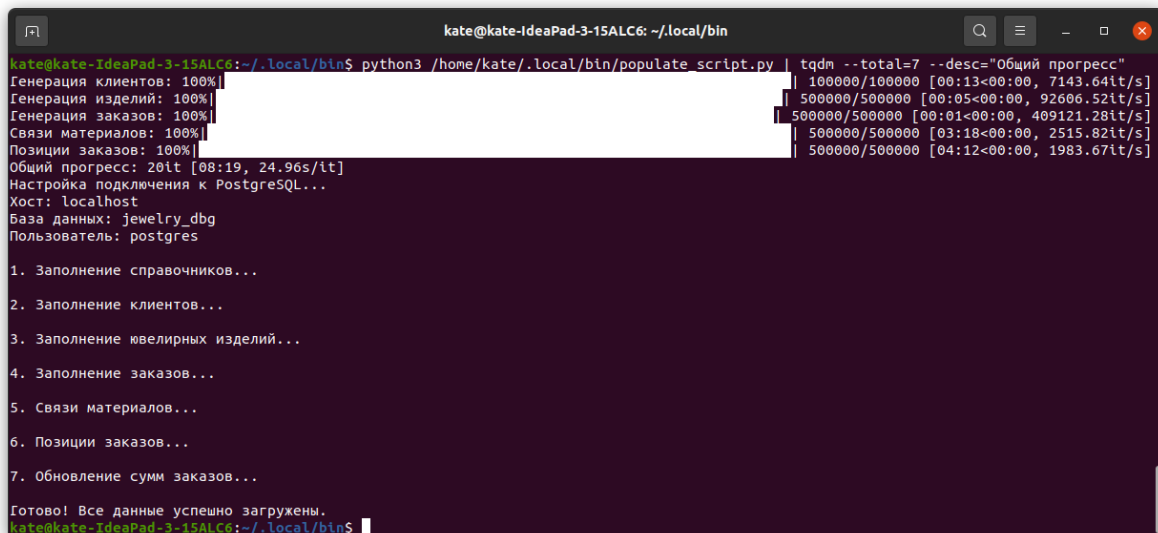
kate@kate-IdeaPad-3-15ALC6:~$ sudo -u postgres psql -c "SHOW shared_buffers;"
Password for user postgres:
psql: error: FATAL: password authentication failed for user "postgres"
kate@kate-IdeaPad-3-15ALC6:~$ sudo -u postgres psql -c "SHOW shared_buffers;"
Password for user postgres:
shared_buffers
-----
4GB
(1 row)

kate@kate-IdeaPad-3-15ALC6:~$
```

Для генерации 500 000 записей в таблицах базы данных ювелирного магазина будем использовать скрипт на языке Python с библиотеками Faker, psycopg2 (для работы с PostgreSQL) и пакетную вставку.

Установим зависимости: `pip install faker psycopg2-binary`

Запустим скрипт `populate_script.py`:

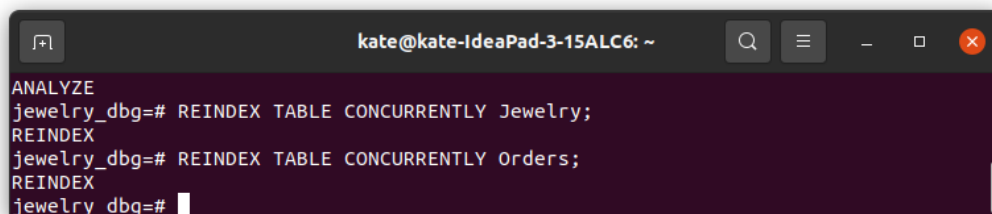


```
kate@kate-IdeaPad-3-15ALC6: ~/.local/bin$ python3 /home/kate/.local/bin/populate_script.py | tqdm --total=7 --desc="Общий прогресс"
Генерация клиентов: 100% | 100000/100000 [00:13<00:00, 7143.64it/s]
Генерация изделий: 100% | 500000/500000 [00:05<00:00, 92606.52it/s]
Генерация заказов: 100% | 500000/500000 [00:01<00:00, 409121.28it/s]
Связи материалов: 100% | 500000/500000 [03:18<00:00, 2515.82it/s]
Позиции заказов: 100% | 500000/500000 [04:12<00:00, 1983.67it/s]
Общий прогресс: 20it [08:19, 24.96s/it]
Настройка подключения к PostgreSQL...
Хост: localhost
База данных: jewelry_dbg
Пользователь: postgres

1. Заполнение справочников...
2. Заполнение клиентов...
3. Заполнение ювелирных изделий...
4. Заполнение заказов...
5. Связи материалов...
6. Позиции заказов...
7. Обновление сумм заказов...

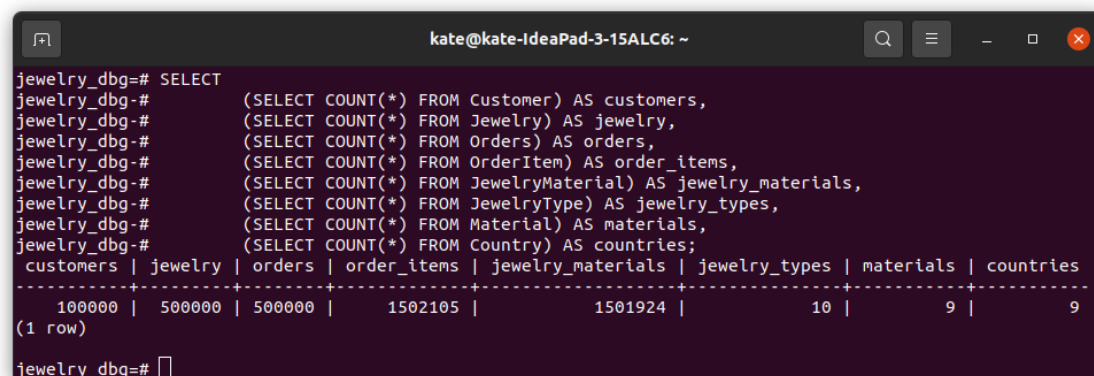
Готово! Все данные успешно загружены.
kate@kate-IdeaPad-3-15ALC6: ~/.local/bin$
```

После загрузки данных выполним ANALYZE и REINDEX.



```
kate@kate-IdeaPad-3-15ALC6: ~$
ANALYZE
jewelry_dbg=# REINDEX TABLE CONCURRENTLY Jewelry;
REINDEX
jewelry_dbg=# REINDEX TABLE CONCURRENTLY Orders;
REINDEX
jewelry_dbg=#
```

Проверим таблицы с данными:



```
kate@kate-IdeaPad-3-15ALC6: ~$
jewelry_dbg=# SELECT
jewelry_dbg-# (SELECT COUNT(*) FROM Customer) AS customers,
jewelry_dbg-# (SELECT COUNT(*) FROM Jewelry) AS jewelry,
jewelry_dbg-# (SELECT COUNT(*) FROM Orders) AS orders,
jewelry_dbg-# (SELECT COUNT(*) FROM OrderItem) AS order_items,
jewelry_dbg-# (SELECT COUNT(*) FROM JewelryMaterial) AS jewelry_materials,
jewelry_dbg-# (SELECT COUNT(*) FROM JewelryType) AS jewelry_types,
jewelry_dbg-# (SELECT COUNT(*) FROM Material) AS materials,
jewelry_dbg-# (SELECT COUNT(*) FROM Country) AS countries;
 customers | jewelry | orders | order_items | jewelry_materials | jewelry_types | materials | countries
-----+-----+-----+-----+-----+-----+-----+-----
 100000 | 500000 | 500000 | 1502105 | 1501924 | 10 | 9 | 9
(1 row)
jewelry_dbg=#
```

## 1.2 SQL SELECT JOIN запрос для 3х таблиц с условием WHERE

```
kate@kate-IdeaPad-3-15ALC6: ~  
jewelry_dbg=#  
jewelry_dbg=# SELECT  
jewelry_dbg=#     j.id AS jewelry_id,  
jewelry_dbg=#     j.name AS jewelry_name,  
jewelry_dbg=#     j.price,  
jewelry_dbg=#     j.weight,  
jewelry_dbg=#     m.name AS material_name,  
jewelry_dbg=#     m.type AS material_type,  
jewelry_dbg=#     m.carat  
jewelry_dbg=# FROM  
jewelry_dbg=#     Jewelry j  
jewelry_dbg=# JOIN  
jewelry_dbg=#     JewelryMaterial jm ON j.id = jm.jewelry_id  
jewelry_dbg=# JOIN  
jewelry_dbg=#     Material m ON jm.material_id = m.id  
jewelry_dbg=# WHERE  
jewelry_dbg=#     j.price BETWEEN 500 AND 5000  
jewelry_dbg=#     AND m.type = 'gemstone'  
jewelry_dbg=#     AND j.weight > 10  
jewelry_dbg=#     AND j.country_id IN (  
jewelry_dbg=#         SELECT id FROM Country  
jewelry_dbg=#         WHERE name IN ('Россия', 'Италия', 'Франция')  
jewelry_dbg=#     )  
jewelry_dbg=# ORDER BY  
jewelry_dbg=#     j.price DESC  
jewelry_dbg=# LIMIT 20;  
jewelry_id | jewelry_name | price | weight | material_name | material_type | carat  
-----  
82506 | Серебряное браслет с сапфиром | 4999.88 | 41.89 | Сапфир | gemstone | 0.80  
130081 | Золотое цепочка с сапфиром | 4999.87 | 23.96 | Изумруд | gemstone | 0.50  
251021 | Золотое подвеска с бриллиантами | 4999.84 | 54.70 | Аметист | gemstone | 0.30  
251021 | Золотое подвеска с бриллиантами | 4999.84 | 54.70 | Рубин | gemstone | 0.60  
251021 | Золотое подвеска с бриллиантами | 4999.84 | 54.70 | Бриллиант | gemstone | 1.00  
499775 | Золотое кольцо с сапфиром | 4999.83 | 79.31 | Сапфир | gemstone | 0.80  
499775 | Золотое кольцо с сапфиром | 4999.83 | 79.31 | Аметист | gemstone | 0.30  
69238 | Платиновое кольцо с сапфиром | 4999.68 | 13.98 | Бриллиант | gemstone | 1.00  
69238 | Платиновое кольцо с сапфиром | 4999.68 | 13.98 | Изумруд | gemstone | 0.50  
69238 | Платиновое кольцо с сапфиром | 4999.68 | 13.98 | Аметист | gemstone | 0.30  
475078 | Платиновое цепочка с бриллиантами | 4999.53 | 87.95 | Бриллиант | gemstone | 1.00  
475078 | Платиновое цепочка с бриллиантами | 4999.53 | 87.95 | Рубин | gemstone | 0.60  
227068 | Серебряное серьги с бриллиантами | 4999.52 | 81.72 | Сапфир | gemstone | 0.80  
227068 | Серебряное серьги с бриллиантами | 4999.52 | 81.72 | Изумруд | gemstone | 0.50  
227068 | Серебряное серьги с бриллиантами | 4999.52 | 81.72 | Бриллиант | gemstone | 1.00  
398564 | Платиновое серьги с бриллиантами | 4999.36 | 82.90 | Изумруд | gemstone | 0.50  
398564 | Платиновое серьги с бриллиантами | 4999.36 | 82.90 | Аметист | gemstone | 0.30  
398564 | Платиновое серьги с бриллиантами | 4999.36 | 82.90 | Рубин | gemstone | 0.60  
93859 | Серебряное цепочка классическое | 4999.31 | 36.74 | Бриллиант | gemstone | 1.00  
214758 | Платиновое серьги с бриллиантами | 4999.18 | 71.45 | Бриллиант | gemstone | 1.00  
(20 rows)  
jewelry_dbg=#
```

Запрос возвращает:

- 20 самых дорогих ювелирных изделий
- Сделанных из драгоценных камней
- Весом более 10 единиц
- Произведенных в России, Италии или Франции
- С ценой от 500 до 5000 денежных единиц
- С подробной информацией о материалах

Выполним запрос с EXPLAIN ANALYZE:

```

kate@kate-IdeaPad-3-15ALC6: ~
QUERY PLAN
-----
Limit (cost=30284.63..30286.96 rows=20 width=100) (actual time=2281.977..2294.134 rows=20 loops=1)
-> Gather Merge (cost=30284.63..41502.68 rows=96148 width=100) (actual time=2281.972..2294.073 rows=20 loops=1)
    Workers Planned: 2
    Workers Launched: 2
    -> Sort (cost=29284.61..29404.79 rows=48074 width=100) (actual time=2275.403..2275.453 rows=16 loops=3)
        Sort Key: j.price DESC
        Sort Method: top-N heapsort Memory: 31kB
        Worker 0: Sort Method: top-N heapsort Memory: 29kB
        Worker 1: Sort Method: top-N heapsort Memory: 29kB
    -> Hash Join (cost=12059.94..28005.37 rows=48074 width=100) (actual time=322.712..2219.587 rows=34359 loops=3)
        Hash Cond: (jm.material_id = m.id)
        -> Parallel Hash Join (cost=12058.76..27670.09 rows=86532 width=77) (actual time=322.435..2053.566 rows=68749 loops=3)
            Hash Cond: (jm.jewelry_id = j.id)
            -> Parallel Seq Scan on jewelrymaterial jm (cost=0.00..12984.02 rows=625802 width=8) (actual time=0.015..809.800 rows=508641 loops=3)
            -> Parallel Hash (cost=11698.67..11698.67 rows=28807 width=73) (actual time=321.010..321.023 rows=22926 loops=3)
                Buckets: 131072 Batches: 1 Memory Usage: 8544kB
                -> Hash Join (cost=1.16..11698.67 rows=28807 width=73) (actual time=0.120..279.787 rows=22926 loops=3)
                    Hash Cond: (j.country_id = country.id)
                    -> Parallel Seq Scan on jewelry j (cost=0.00..11363.83 rows=86422 width=77) (actual time=0.024..127.960 rows=68951 loops=3)
                        Filter: ((price >= '500'::numeric) AND (price <= '5000'::numeric) AND (weight > '10'::numeric))
                        Rows Removed by Filter: 97716
                    -> Seq Scan on country (cost=0.00..1.12 rows=3 width=4) (actual time=0.031..0.041 rows=3 loops=3)
                        Buckets: 1024 Batches: 1 Memory Usage: 9kB
                        Filter: ((name)::text = ANY ('(Россия,Италия,Франция)::text[]))
                        Rows Removed by Filter: 6
                -> Hash (cost=1.11..1.11 rows=5 width=31) (actual time=0.060..0.064 rows=5 loops=3)
                    Buckets: 1024 Batches: 1 Memory Usage: 9kB
                    -> Seq Scan on material m (cost=0.00..1.11 rows=5 width=31) (actual time=0.029..0.041 rows=5 loops=3)
                        Filter: ((type)::text = 'gemstone'::text)
                        Rows Removed by Filter: 4
        Planning Time: 0.981 ms
        Execution Time: 2294.231 ms
(33 rows)

jewelry_dbg=#
```

## 2. Добавление индексов на поля, используемые в WHERE

```

kate@kate-IdeaPad-3-15ALC6: ~
jewelry_dbg=# CREATE INDEX idx_jewelry_price_weight ON Jewelry(price, weight);
CREATE INDEX
jewelry_dbg=# CREATE INDEX idx_jewelry_country_id ON Jewelry(country_id);
CREATE INDEX
jewelry_dbg=# CREATE INDEX idx_material_type ON Material(type);
CREATE INDEX
jewelry_dbg=# ANALYZE Jewelry;
ANALYZE
jewelry_dbg=# ANALYZE Material;
ANALYZE
jewelry_dbg=#
```

```

kate@kate-IdeaPad-3-15ALC6: ~
QUERY PLAN
-----
Limit (cost=0.85..20.35 rows=20 width=100) (actual time=0.544..2.386 rows=20 loops=1)
-> Nested Loop (cost=0.85..111754.73 rows=114627 width=100) (actual time=0.539..2.310 rows=20 loops=1)
    Join Filter: (jm.material_id = m.id)
    Rows Removed by Join Filter: 167
    -> Nested Loop (cost=0.85..97425.26 rows=206328 width=77) (actual time=0.187..1.361 rows=46 loops=1)
        -> Nested Loop (cost=0.42..48035.71 rows=68688 width=73) (actual time=0.166..0.881 rows=15 loops=1)
            Join Filter: (j.country_id = country.id)
            Rows Removed by Join Filter: 85
            -> Index Scan Backward using idx_jewelry_price_weight on jewelry j (cost=0.42..39276.86 rows=206064 width=77) (actual time=0.049..0.274 rows=38 loops=1)
                Index Cond: ((price >= '500'::numeric) AND (price <= '5000'::numeric) AND (weight > '10'::numeric))
            -> Materialize (cost=0.00..1.14 rows=3 width=4) (actual time=0.002..0.007 rows=3 loops=38)
                -> Seq Scan on country (cost=0.00..1.12 rows=3 width=4) (actual time=0.011..0.023 rows=3 loops=1)
                    Filter: ((name)::text = ANY ('(Россия,Италия,Франция)::text[]))
                    Rows Removed by Filter: 6
                -> Index Only Scan using jewelrymaterial_pkey on jewelrymaterial jm (cost=0.43..0.69 rows=3 width=8) (actual time=0.009..0.016 rows=3 loops=15)
                    Index Cond: (jewelry_id = j.id)
                    Heap Fetches: 46
        -> Materialize (cost=0.00..1.14 rows=5 width=31) (actual time=0.002..0.009 rows=4 loops=46)
            -> Seq Scan on material m (cost=0.00..1.11 rows=5 width=31) (actual time=0.008..0.021 rows=5 loops=1)
                Filter: ((type)::text = 'gemstone'::text)
                Rows Removed by Filter: 4
    Planning Time: 1.322 ms
    Execution Time: 2.482 ms
(23 rows)

(END)
```

## 3. Замена типа одного из индексов на bitmap



PostgreSQL автоматически использует битмап-сканирование при комбинации индексов.

```
kate@kate-IdeaPad-3-15ALC6: ~/.local/bin
jewelry_dbg=# CREATE INDEX idx_jewelry_price_bitmap ON Jewelry USING btree (price);
CREATE INDEX
jewelry_dbg=#
```

```
QUERY PLAN
-----
Limit  (cost=0.85..20.18 rows=20 width=100) (actual time=0.495..2.365 rows=20 loops=1)
-> Nested Loop  (cost=0.85..111145.86 rows=115001 width=100) (actual time=0.490..2.287 rows=20 loops=1)
    Join Filter: (jm.material_id = m.id)
    Rows Removed by Join Filter: 167
    -> Nested Loop  (cost=0.85..96769.59 rows=207002 width=77) (actual time=0.204..1.332 rows=46 loops=1)
        Join Filter: (j.country_id = country.id)
        Rows Removed by Join Filter: 85
        -> Index Scan Backward using idx_jewelry_price_bitmap on jewelry j  (cost=0.42..38478.42 rows=206737 width=77) (actual time=0.061..0.230 rows=38 loops=1)
            Index Cond: ((price >= '500'::numeric) AND (price <= '5000'::numeric))
            Filter: (weight > '10'::numeric)
            Rows Removed by Filter: 7
        -> Materialize  (cost=0.00..1.14 rows=3 width=4) (actual time=0.002..0.007 rows=3 loops=38)
            -> Seq Scan on country  (cost=0.00..1.12 rows=3 width=4) (actual time=0.012..0.022 rows=3 loops=1)
                Filter: ((name)::text = ANY ('{Россия,Италия,Франция}'::text[]))
                Rows Removed by Filter: 0
            -> Index Only Scan using jewelrymaterial_pkey on jewelrymaterial jm  (cost=0.43..0.69 rows=3 width=8) (actual time=0.010..0.017 rows=3 loops=15)
                Index Cond: (jewelry_id = j.id)
                Heap Fetches: 40
    -> Materialize  (cost=0.00..1.14 rows=5 width=31) (actual time=0.002..0.009 rows=4 loops=46)
        -> Seq Scan on material m  (cost=0.00..1.11 rows=5 width=31) (actual time=0.007..0.020 rows=5 loops=1)
            Filter: ((type)::text = 'gemstone'::text)
            Rows Removed by Filter: 4
Planning Time: 1.126 ms
Execution Time: 2.476 ms
(25 rows)
:
```

#### 4. Добавление индексов на ссылочные поля, используемые в JOIN

```
kate@kate-IdeaPad-3-15ALC6: ~
jewelry_dbg=#
jewelry_dbg=# CREATE INDEX idx_jewelrymaterial_jewelry_id ON JewelryMaterial(jewelry_id);
CREATE INDEX
jewelry_dbg=# CREATE INDEX idx_jewelrymaterial_material_id ON JewelryMaterial(material_id);
CREATE INDEX
jewelry_dbg=# ANALYZE Jewelry;
ANALYZE
jewelry_dbg=# ANALYZE Material;
ANALYZE
jewelry_dbg=# ANALYZE JewelryMaterial;
ANALYZE
jewelry_dbg=# ANALYZE Country;
ANALYZE
jewelry_dbg=#
```

```

Limit (cost=0.85..20.61 rows=20 width=100) (actual time=0.514..2.379 rows=20 loops=1)
-> Nested Loop (cost=0.85..114175.31 rows=115542 width=100) (actual time=0.509..2.303 rows=20 loops=1)
    Join Filter: (jm.material_id = m.id)
    Rows Removed by Join Filter: 167
-> Nested Loop (cost=0.85..99731.46 rows=207975 width=77) (actual time=0.177..1.337 rows=46 loops=1)
    -> Nested Loop (cost=0.42..48158.51 rows=69236 width=73) (actual time=0.138..0.841 rows=15 loops=1)
        Join Filter: (j.country_id = country.id)
        Rows Removed by Join Filter: 85
        -> Index Scan Backward using idx_jewelry_price_weight on jewelry j (cost=0.42..39329.75 rows=207709 width=77) (
actual time=0.020..0.244 rows=38 loops=1)
            Index Cond: ((price >= '500'::numeric) AND (price <= '5000'::numeric) AND (weight > '10'::numeric))
        -> Materialize (cost=0.00..1.14 rows=3 width=4) (actual time=0.002..0.007 rows=3 loops=38)
            -> Seq Scan on country (cost=0.00..1.12 rows=3 width=4) (actual time=0.012..0.023 rows=3 loops=1)
                Filter: ((name)::text = ANY ('{Россия,Италия,Франция}'::text[]))
                Rows Removed by Filter: 6
            -> Index Only Scan using jewelrmaterial_pkey on jewelrmaterial jm (cost=0.43..0.70 rows=4 width=8) (actual time=0.0
11..0.018 rows=3 loops=15)
                Index Cond: (jewelry_id = j.id)
                Heap Fetches: 46
        -> Materialize (cost=0.00..1.14 rows=5 width=31) (actual time=0.002..0.009 rows=4 loops=46)
            -> Seq Scan on material m (cost=0.00..1.11 rows=5 width=31) (actual time=0.007..0.020 rows=5 loops=1)
                Filter: ((type)::text = 'gemstone'::text)
                Rows Removed by Filter: 4
Planning Time: 1.279 ms
Execution Time: 2.472 ms
~
~
(END)

```

## 5. QuarryPlan. Заполнение сравнительной таблицы

Индексы	Производительность (в секундах)	Глобальные ссылки
нет	2294.231 ms	jm.material_id = m.id jm.jewelry_id = j.id j.country_id = country.id
idx_jewelry_price_weight idx_jewelry_country_id idx_material_type	2.482 ms	jm.material_id = m.id jewelry_id = j.id j.country_id = country.id
idx_jewelry_price_weight idx_jewelry_country_id idx_material_type idx_jewelry_price_bitmap SET enable_bitmapscan = on;	2.476 ms	jm.material_id = m.id jewelry_id = j.id j.country_id = country.id

idx_jewelry_price_weight idx_jewelry_country_id idx_material_type idx_jewelrmaterial_jewelry_id idx_jewelrmaterial_material_id	2.472 ms	jm.material_id = m.id jewelry_id = j.id j.country_id = country.id
--	----------	---

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Генерация фейковых данных Faker. Python package - [Эл.ресурс]  
<https://github.com/joke2k/faker>
2. Документация к PostgreSQL 17.4 - [Эл.ресурс]  
<https://postgrespro.ru/docs/postgresql/17/index>
3. Наполнение базы данных - [Эл.ресурс]  
<https://postgrespro.ru/docs/postgresql/9.6/populate>
4. Psycopg – PostgreSQL database adapter for Python - [Эл.ресурс]  
<https://www.psycopg.org/docs/>