Задание 1: BRIN индексы и bitmap-сканирование

1. Удалите старую базу данных, если есть:

```
docker compose down
```

2. Поднимите базу данных из src/docker-compose.yml:

```
docker compose down && docker compose up -d
```

3. Обновите статистику:

```
ANALYZE t_books;
```

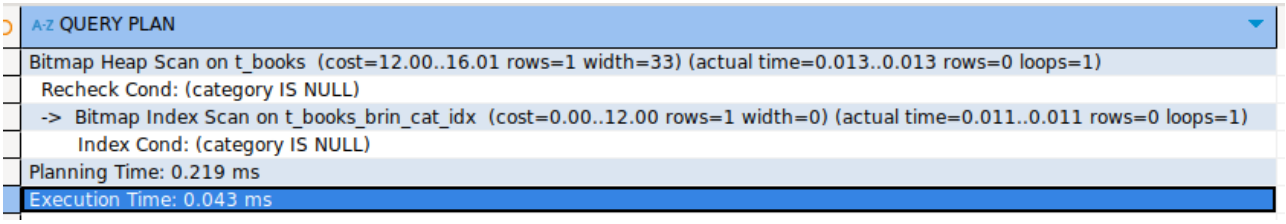
4. Создайте BRIN индекс по колонке category:

```
CREATE INDEX t_books_brin_cat_idx ON t_books USING brin(category);
```

5. Найдите книги с NULL значением category:

```
EXPLAIN ANALYZE
SELECT * FROM t_books WHERE category IS NULL;
```

План выполнения:



Объясните результат:

Из-за того, что мы использовали brin индекс, планировщик решил, использовать его для проверки диапозона строк где `category is null` таких не оказалось, следовательно, запрос очень быстро отработал

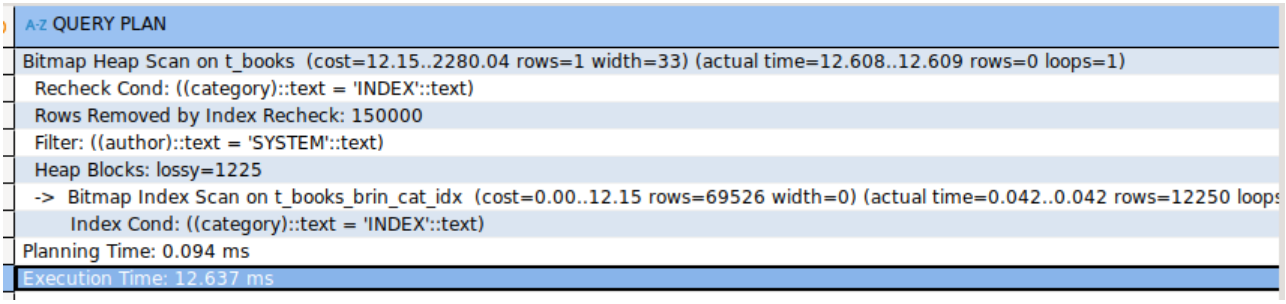
6. Создайте BRIN индекс по автору:

```
CREATE INDEX t_books_brin_author_idx ON t_books USING brin(author);
```

7. Выполните поиск по категории и автору:

```
EXPLAIN ANALYZE
SELECT * FROM t_books
WHERE category = 'INDEX' AND author = 'SYSTEM';
```

План выполнения:



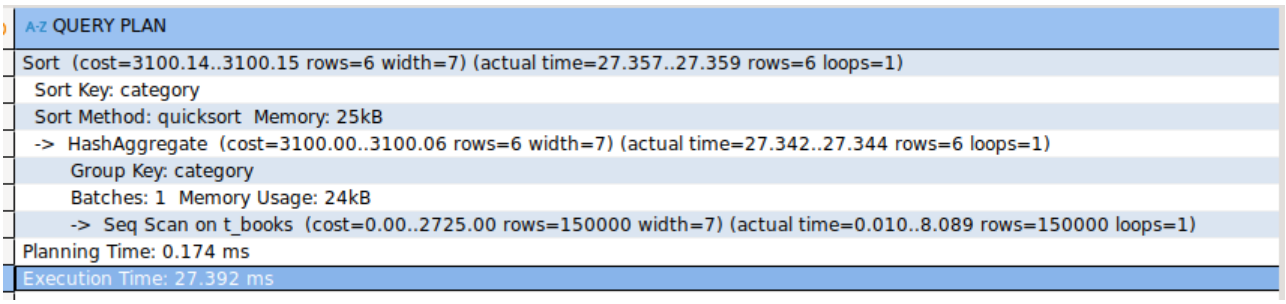
Объясните результат (обратите внимание на bitmap scan):

Использовался индекс по `category` потому что он имеет более равномерное распределение (как мы видим мы просто генерим категории) из-за этого лучше его использовать, тк будет отсечено много блоков данных. `bitmap scan` здесь использовался, потому что планировщик решил тратить меньше на чтение данных, сканируя только те блоки, где может содержаться нужная строка

8. Получите список уникальных категорий:

```
EXPLAIN ANALYZE
SELECT DISTINCT category
FROM t_books
ORDER BY category;
```

План выполнения:



Объясните результат:

Тк нет индексов мы видим `seq scan`, `hash aggregate` используется для того, чтобы выбрать уникальные значения

9. Подсчитайте книги, где автор начинается на 'S':

```
EXPLAIN ANALYZE
SELECT COUNT(*)
FROM t_books
WHERE author LIKE 'S%';
```

План выполнения:

QUERY PLAN
Aggregate (cost=3100.03..3100.05 rows=1 width=8) (actual time=11.315..11.317 rows=1 loops=1)
-> Seq Scan on t_books (cost=0.00..3100.00 rows=14 width=0) (actual time=11.309..11.310 rows=0 loops=1)
Filter: ((author)::text ~~ 'S%'::text)
Rows Removed by Filter: 150000
Planning Time: 0.343 ms
Execution Time: 11.347 ms

Объясните результат:

здесь используется текстовый паттерн, который `brin` индекс по `author` не может обработать, тк он индексирует не сами записи, а только агрегированные статистики

10. Создайте индекс для регистронезависимого поиска:

```
CREATE INDEX t_books_lower_title_idx ON t_books(LOWER(title));
```

11. Подсчитайте книги, начинающиеся на 'O':

```
EXPLAIN ANALYZE
SELECT COUNT(*)
FROM t_books
WHERE LOWER(title) LIKE 'o%';
```

План выполнения:

QUERY PLAN
Aggregate (cost=3476.88..3476.89 rows=1 width=8) (actual time=27.415..27.416 rows=1 loops=1)
-> Seq Scan on t_books (cost=0.00..3475.00 rows=750 width=0) (actual time=27.406..27.408 rows=1 loops=1)
Filter: (lower((title)::text) ~~ 'o%'::text)
Rows Removed by Filter: 149999
Planning Time: 0.108 ms
Execution Time: 27.445 ms

Объясните результат:

`B-tree` индекс не использовался, тк у нас записи только начинающиеся на `author` и планировщик это понял, поэтому прошелся `seq scan`

12. Удалите созданные индексы:

```
DROP INDEX t_books_brin_cat_idx;
DROP INDEX t_books_brin_author_idx;
DROP INDEX t_books_lower_title_idx;
```

13. Создайте составной BRIN индекс:

```
CREATE INDEX t_books_brin_cat_auth_idx ON t_books
USING brin(category, author);
```

14. Повторите запрос из шага 7:

```
EXPLAIN ANALYZE
SELECT * FROM t_books
WHERE category = 'INDEX' AND author = 'SYSTEM';
```

План выполнения:

QUERY PLAN
Bitmap Heap Scan on t_books (cost=12.15..2280.04 rows=1 width=33) (actual time=0.685..0.686 rows=0 loops=1)
Recheck Cond: (((category)::text = 'INDEX'::text) AND ((author)::text = 'SYSTEM'::text))
Rows Removed by Index Recheck: 8837
Heap Blocks: lossy=73
-> Bitmap Index Scan on t_books_brin_cat_auth_idx (cost=0.00..12.15 rows=69526 width=0) (actual time=0.015..0.016 rows=
Index Cond: (((category)::text = 'INDEX'::text) AND ((author)::text = 'SYSTEM'::text))
Planning Time: 0.127 ms
Execution Time: 0.702 ms

Объясните результат:

Использовался индекс которые мы создали, тк тут точное соответствие его специфике - то есть поиск по `category` и `author`