Гетманов Михаил БПИ223

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:

```
Bitmap Heap Scan on t_books  (cost=12.00..16.01 rows=1 width=33) (actual
time=0.012..0.013 rows=0 loops=1)
Recheck Cond: (category IS NULL)
->  Bitmap Index Scan on t_books_brin_cat_idx  (cost=0.00..12.00 rows=1
width=0) (actual time=0.011..0.011 rows=0 loops=1)
      Index Cond: (category IS NULL)
Planning Time: 0.080 ms
Execution Time: 0.034 ms
```

Объясните результат:

- Используется BitMap
- Запрос стоит от 12 до 16.01
- планируемое время на запрос - 0.080 мс
- фактическое время на запрос - 0.034 мс

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:

```
Bitmap Heap Scan on t_books  (cost=12.00..16.02 rows=1 width=33) (actual
time=14.893..14.895 rows=0 loops=1)
Recheck Cond: ((category)::text = 'INDEX'::text)
Rows Removed by Index Recheck: 150000
Filter: ((author)::text = 'SYSTEM'::text)
Heap Blocks: lossy=1225
->  Bitmap Index Scan on t_books_brin_cat_idx  (cost=0.00..12.00 rows=1
width=0) (actual time=0.044..0.045 rows=12250 loops=1)
      Index Cond: ((category)::text = 'INDEX'::text)
Planning Time: 0.155 ms
Execution Time: 14.915 ms
```

Объясните результат (обратите внимание на bitmap scan):

- Используется BitMap
- Запрос стоит от 12 до 16.02
- планируемое время на запрос - 0.155 мс
- фактическое время на запрос - 14.915 мс
- затраченное время увеличилось из-за увеличения сложности запроса

8. Получите список уникальных категорий:

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

План выполнения:

```
Sort  (cost=3100.11..3100.12 rows=5 width=7) (actual time=29.564..29.567
rows=6 loops=1)
Sort Key: category
Sort Method: quicksort  Memory: 25kB
->  HashAggregate  (cost=3100.00..3100.05 rows=5 width=7) (actual
time=29.541..29.544 rows=6 loops=1)
      Group Key: category
      Batches: 1  Memory Usage: 24kB
      ->  Seq Scan on t_books  (cost=0.00..2725.00 rows=150000 width=7)
(actual time=0.005..7.662 rows=150000 loops=1)
Planning Time: 0.079 ms
Execution Time: 29.604 ms
```

Объясните результат:

- используется для сортировки QuickSort
- Запрос стоит от 3100.11 до 3100.12
- планируемое время на запрос - 0.079 мс
- фактическое время на запрос - 29.604 мс

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

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

План выполнения:

```
Aggregate  (cost=3100.04..3100.05 rows=1 width=8) (actual
time=13.027..13.028 rows=1 loops=1)
->  Seq Scan on t_books  (cost=0.00..3100.00 rows=15 width=0) (actual
time=13.023..13.023 rows=0 loops=1)
      Filter: ((author)::text ~~ 'S%'::text)
      Rows Removed by Filter: 150000
Planning Time: 0.108 ms
Execution Time: 13.057 ms
```

Объясните результат:

- Запрос выполняет агрегацию
- идёт последовательный анализ строк таблицы
- Запрос стоит от 3100.04 до 3100.05
- планируемое время на запрос - 0.108 мс
- фактическое время на запрос - 13.057 мс

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:

```
Aggregate  (cost=3476.88..3476.89 rows=1 width=8) (actual time=42.559..42.561
rows=1 loops=1)
->  Seq Scan on t_books  (cost=0.00..3475.00 rows=750 width=0) (actual
time=42.551..42.553 rows=1 loops=1)
      Filter: (lower((title)::text) ~~ 'o%'::text)
      Rows Removed by Filter: 149999
Planning Time: 0.067 ms
Execution Time: 42.583 ms
```

Объясните результат:

- Запрос выполняет агрегацию
- идёт последовательный анализ строк таблицы
- Запрос стоит от 3476.88 до 3476.89
- планируемое время на запрос - 0.067 мс

- фактическое время на запрос - 42.583 мс

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:

```
Bitmap Heap Scan on t_books  (cost=12.00..16.02 rows=1 width=33) (actual
time=0.987..0.988 rows=0 loops=1)
Recheck Cond: (((category)::text = 'INDEX'::text) AND ((author)::text =
'SYSTEM'::text))
Rows Removed by Index Recheck: 8844
Heap Blocks: lossy=73
->  Bitmap Index Scan on t_books_brin_cat_auth_idx  (cost=0.00..12.00 rows=1
width=0) (actual time=0.021..0.022 rows=730 loops=1)
      Index Cond: (((category)::text = 'INDEX'::text) AND ((author)::text =
'SYSTEM'::text))
Planning Time: 0.211 ms
Execution Time: 1.018 ms
```

Объясните результат:

- Используется BitMap
- Запрос стоит от 12 до 16.02
- планируемое время на запрос - 0.211 мс
- фактическое время на запрос - 1.018 мс

Задание 2

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 полнотекстовый индекс:

```
CREATE INDEX t_books_fts_idx ON t_books
USING GIN (to_tsvector('english', title));
```

5. Найдите книги, содержащие слово 'expert':

```
EXPLAIN ANALYZE
SELECT * FROM t_books
WHERE to_tsvector('english', title) @@ to_tsquery('english', 'expert');
```

План выполнения:

```
Bitmap Heap Scan on t_books  (cost=21.03..1335.59 rows=750 width=33)
(actual time=0.019..0.020 rows=1 loops=1)
 "  Recheck Cond: (to_tsvector('english'::regconfig, (title)::text) @@
'''expert'''::tsquery)"
 Heap Blocks: exact=1
 ->  Bitmap Index Scan on t_books_fts_idx  (cost=0.00..20.84 rows=750
width=0) (actual time=0.014..0.015 rows=1 loops=1)
 "        Index Cond: (to_tsvector('english'::regconfig, (title)::text)
@@ '''expert'''::tsquery)"
 Planning Time: 0.375 ms
 Execution Time: 0.045 ms
```

Объясните результат:

- используется BitMap для поиска
- запрос стоит от 21.03 до 1335.59
- планируемое время запроса - 0.375 ms
- фактическое время запроса - 0.045 ms

6. Удалите индекс:

```
DROP INDEX t_books_fts_idx;
```

7. Создайте таблицу lookup:

```
CREATE TABLE t_lookup (
     item_key VARCHAR(10) NOT NULL,
     item_value VARCHAR(100)
);
```

8. Добавьте первичный ключ:

```
ALTER TABLE t_lookup
ADD CONSTRAINT t_lookup_pk PRIMARY KEY (item_key);
```

9. Заполните данными:

```
INSERT INTO t_lookup
SELECT
     LPAD(CAST(generate_series(1, 150000) AS TEXT), 10, '0'),
     'Value_' || generate_series(1, 150000);
```

10. Создайте кластеризованную таблицу:

```
CREATE TABLE t_lookup_clustered (
     item_key VARCHAR(10) PRIMARY KEY,
     item_value VARCHAR(100)
);
```

11. Заполните её теми же данными:

```
INSERT INTO t_lookup_clustered
SELECT * FROM t_lookup;

CLUSTER t_lookup_clustered USING t_lookup_clustered_pkey;
```

12. Обновите статистику:

```
ANALYZE t_lookup;
ANALYZE t_lookup_clustered;
```

13. Выполните поиск по ключу в обычной таблице:

```
EXPLAIN ANALYZE
SELECT * FROM t_lookup WHERE item_key = '0000000455';
```

План выполнения:

```
Index Scan using t_lookup_pk on t_lookup  (cost=0.42..8.44 rows=1
width=23) (actual time=0.014..0.014 rows=1 loops=1)
Index Cond: ((item_key)::text = '0000000455'::text)
Planning Time: 0.202 ms
Execution Time: 0.026 ms
```

Объясните результат:

- используется поиск по индексу
- запрос стоит от 0.42 до 8.44
- планируемое время запроса - 0.202 ms
- фактическое время запроса - 0.026 ms

14. Выполните поиск по ключу в кластеризованной таблице:

```
EXPLAIN ANALYZE
SELECT * FROM t_lookup_clustered WHERE item_key = '0000000455';
```

План выполнения:

```
Index Scan using t_lookup_clustered_pkey on t_lookup_clustered
(cost=0.42..8.44 rows=1 width=23) (actual time=0.030..0.031 rows=1
loops=1)
Index Cond: ((item_key)::text = '0000000455'::text)
Planning Time: 0.124 ms
Execution Time: 0.044 ms
```

Объясните результат:

- используется поиск по индексу
- запрос стоит от 0.42 до 8.44
- планируемое время запроса - 0.124 ms
- фактическое время запроса - 0.044 ms

15. Создайте индекс по значению для обычной таблицы:

```
CREATE INDEX t_lookup_value_idx ON t_lookup(item_value);
```

16. Создайте индекс по значению для кластеризованной таблицы:

```
CREATE INDEX t_lookup_clustered_value_idx
ON t_lookup_clustered(item_value);
```

17. Выполните поиск по значению в обычной таблице:

```
EXPLAIN ANALYZE
SELECT * FROM t_lookup WHERE item_value = 'T_BOOKS';
```

План выполнения:

```
Index Scan using t_lookup_value_idx on t_lookup  (cost=0.42..8.44 rows=1
width=23) (actual time=0.016..0.016 rows=0 loops=1)
Index Cond: ((item_value)::text = 'T_BOOKS'::text)
Planning Time: 0.312 ms
Execution Time: 0.030 ms
```

Объясните результат:

- используется поиск по индексу
- запрос стоит от 0.42 до 8.44
- планируемое время запроса - 0.312 ms
- фактическое время запроса - 0.030 ms

18. Выполните поиск по значению в кластеризованной таблице:

```
EXPLAIN ANALYZE
SELECT * FROM t_lookup_clustered WHERE item_value = 'T_BOOKS';
```

План выполнения:

```
Index Scan using t_lookup_clustered_value_idx on t_lookup_clustered
(cost=0.42..8.44 rows=1 width=23) (actual time=0.014..0.014 rows=0
loops=1)
Index Cond: ((item_value)::text = 'T_BOOKS'::text)
Planning Time: 0.188 ms
Execution Time: 0.025 ms
```

Объясните результат:

- используется поиск по индексу
- запрос стоит от 0.42 до 8.44
- планируемое время запроса - 0.188 ms
- фактическое время запроса - 0.025 ms

19. Сравните производительность поиска по значению в обычной и кластеризованной таблицах:

    Сравнение:

    - поиск в обычной и кластеризованной таблицах занял примерно одинаковое время
    - поиск в обычной таблице согласно плану стоит дороже

Задание 3

20. Создайте таблицу с большим количеством данных:

```
CREATE TABLE test_cluster AS
SELECT
    generate_series(1,1000000) as id,
    CASE WHEN random() < 0.5 THEN 'A' ELSE 'B' END as category,
    md5(random()::text) as data;
```

21. Создайте индекс:

```
CREATE INDEX test_cluster_cat_idx ON test_cluster(category);
```

22. Измерьте производительность до кластеризации:

```
EXPLAIN ANALYZE
SELECT * FROM test_cluster WHERE category = 'A';
```

    План выполнения:

```
Bitmap Heap Scan on test_cluster  (cost=59.17..7696.73 rows=5000
width=68) (actual time=13.386..91.058 rows=500210 loops=1)
Recheck Cond: (category = 'A'::text)
Heap Blocks: exact=8334
->  Bitmap Index Scan on test_cluster_cat_idx  (cost=0.00..57.92
rows=5000 width=0) (actual time=12.286..12.286 rows=500210 loops=1)
        Index Cond: (category = 'A'::text)
Planning Time: 0.253 ms
Execution Time: 104.200 ms
```

    Объясните результат:

    - используется BitMap
    - стоит запрос от 59.17 до 7696.73
    - планируемое время - 0.253 ms
    - фактическое время - 104.200 ms

23. Выполните кластеризацию:

```
CLUSTER test_cluster USING test_cluster_cat_idx;
```

    Результат:

```
[2024-12-09 22:36:42] completed in 833 ms
```

24. Измерьте производительность после кластеризации:

```
EXPLAIN ANALYZE
SELECT * FROM test_cluster WHERE category = 'A';
```

План выполнения:

```
Bitmap Heap Scan on test_cluster  (cost=5564.45..20137.20 rows=499100
width=39) (actual time=10.964..50.887 rows=500210 loops=1)
Recheck Cond: (category = 'A'::text)
Heap Blocks: exact=4169
->  Bitmap Index Scan on test_cluster_cat_idx  (cost=0.00..5439.68
rows=499100 width=0) (actual time=10.464..10.464 rows=500210 loops=1)
        Index Cond: (category = 'A'::text)
Planning Time: 0.187 ms
Execution Time: 64.182 ms
```

Объясните результат:

- используется BitMap
- стоит запрос от 5564.45 до 20137.20
- планируемое время - 0.187 ms
- фактическое время - 64.182 ms

25. Сравните производительность до и после кластеризации:

Сравнение:

- максимальная стоимость запроса увеличилась почти в три раза
- запрос после кластеризации занял почти в два раза меньше