

Национальный исследовательский ядерный университет «МИФИ»

(Московский Инженерно-Физический Институт)

Кафедра №42 «Криптология и кибербезопасность»

Лабораторная работа №2-4

«Индексы»

Тимин Александр Б21-515 (2024г.)

1 Система индексов

Начнем с разработки системы индексов для базы данных.

Сначала определим типичные запросы, которые могут использоваться в нашей базе данных. Основываясь на предоставленной ERD-схеме, определим типичные запросы:

- Получение информации о сериале:
 - Запросы по shows для поиска сериала по названию (title) или году выпуска (year).
 - Фильтрация сериалов по бюджету (budget).
- Работа с актерами:
 - Поиск актеров по имени (name).
 - Выборка всех сериалов, в которых участвовал определенный актер.
- Обработка отзывов:
 - Получение всех отзывов для конкретного сериала (show_id).
 - Фильтрация отзывов по рейтингу (rating).
- Работа с сезонами и эпизодами:
 - Выбор всех сезонов конкретного сериала (show_id).
 - Получение всех эпизодов конкретного сезона (season_id).
- Связи актеров и ролей:
 - Фильтрация по актерам (actor_id) или ролям (role) для определенного сериала (show_id).

Теперь мы можем спроектировать индексы. Для каждого из перечисленных запросов создадим индексы, которые ускорят их выполнение.

Таблица shows:

- Индекс на title для поиска по названию

```
CREATE INDEX idx_shows_title ON  
series.shows(title);
```

- Индекс на year для фильтрации по году

```
CREATE INDEX idx_shows_year ON series.shows(year);
```

Таблица actors:

- Индекс на name для поиска по имени

```
CREATE INDEX idx_actors_name ON  
people.actors (name);
```

Таблица reviews:

- Индекс на show_id для быстрого доступа к отзывам сериала

```
CREATE INDEX idx_reviews_show_id ON  
series.reviews (show_id);
```

- Индекс на rating для фильтрации по рейтингу

```
CREATE INDEX idx_reviews_rating ON  
series.reviews (rating);
```

Таблица seasons:

- Индекс на show_id для получения сезонов сериала

```
CREATE INDEX idx_seasons_show_id ON  
series.seasons (show_id);
```

Таблица episodes:

- Индекс на season_id для получения эпизодов сезона

```
CREATE INDEX idx_episodes_season_id ON  
series.episodes (season_id);
```

Таблица cast:

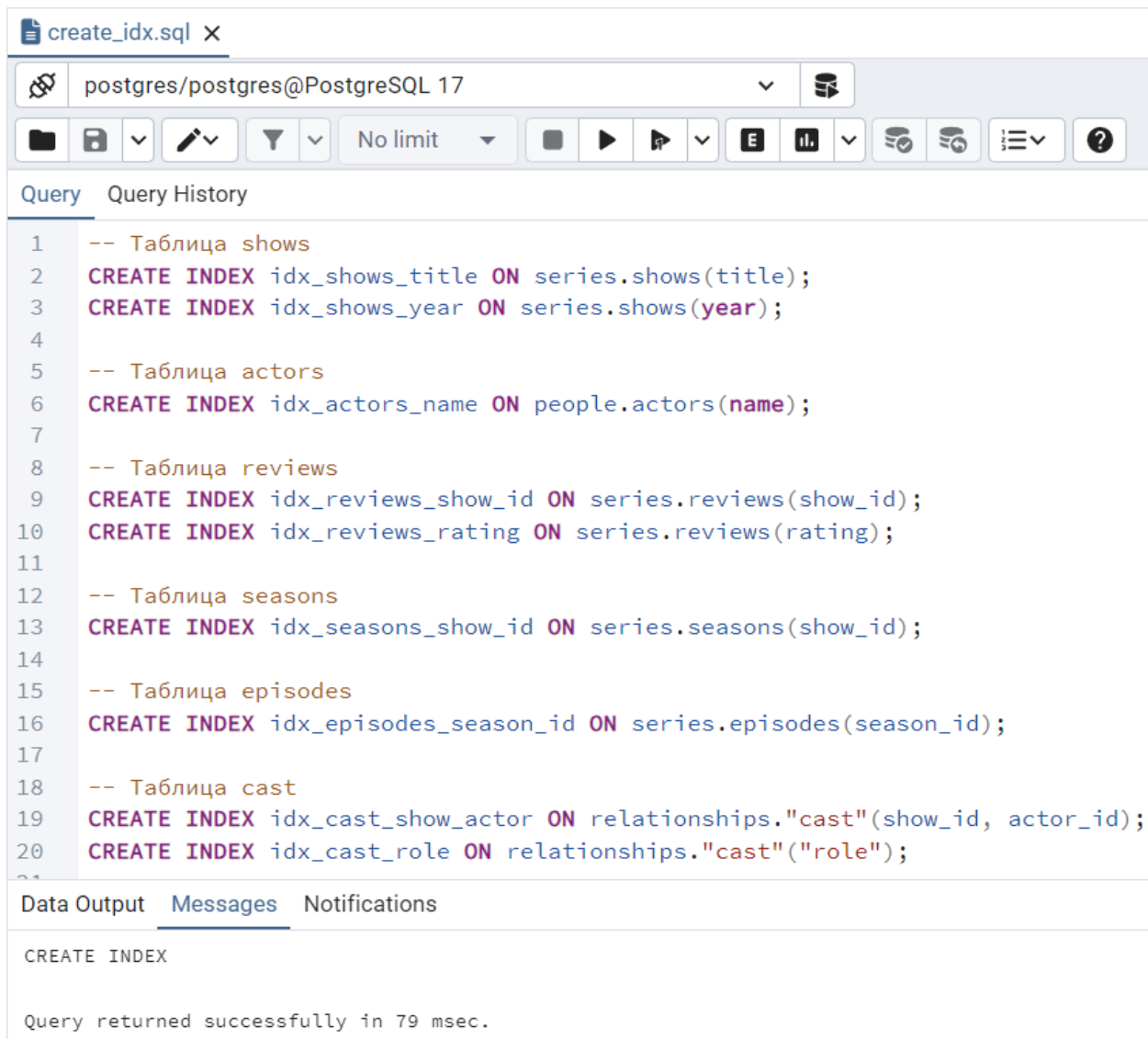
- Составной индекс на show_id и actor_id для связей сериалов и актеров

```
CREATE INDEX idx_cast_show_actor ON  
relationships."cast" (show_id, actor_id);
```

- Индекс на role для фильтрации по ролям

```
CREATE INDEX idx_cast_role ON  
relationships."cast" ("role");
```

Создадим эти индексы.



```
create_idx.sql X
postgres/postgres@PostgreSQL 17
No limit
Query Query History
1  -- Таблица shows
2  CREATE INDEX idx_shows_title ON series.shows(title);
3  CREATE INDEX idx_shows_year ON series.shows(year);
4
5  -- Таблица actors
6  CREATE INDEX idx_actors_name ON people.actors(name);
7
8  -- Таблица reviews
9  CREATE INDEX idx_reviews_show_id ON series.reviews(show_id);
10 CREATE INDEX idx_reviews_rating ON series.reviews(rating);
11
12 -- Таблица seasons
13 CREATE INDEX idx_seasons_show_id ON series.seasons(show_id);
14
15 -- Таблица episodes
16 CREATE INDEX idx_episodes_season_id ON series.episodes(season_id);
17
18 -- Таблица cast
19 CREATE INDEX idx_cast_show_actor ON relationships."cast"(show_id, actor_id);
20 CREATE INDEX idx_cast_role ON relationships."cast"("role");
21
Data Output Messages Notifications
CREATE INDEX
Query returned successfully in 79 msec.
```

2 Статистическая информация

Данный пункт в ТЗ был помечен, как «дополнительное кармическое задание», так как я не верю в карму, решил его не выполнять 😊

3 Много новых данных

С помощью скрипта добавил кучу записей в БД (ссылка на скрипт – в приложении). Запросы не слишком оптимизированы, но я думаю, это не большая проблема.

```
insert.sql x
postgres/postgres@PostgreSQL 17
Query Query History
79 (CURRENT_DATE - (random() * 365 * 5) * interval '1 day')
80 FROM
81 series.seasons se
82 ORDER BY random()
83 LIMIT 1;
84 END LOOP;
85 END $$;
86
87 -- Добавление данных в таблицы reviews
88 DO $$
89 BEGIN
90 FOR i IN 1..1000 LOOP
91 INSERT INTO series.reviews (show_id, reviewer, text, rating)
92 SELECT
93 s.id,
94 'Reviewer ' || i,
95 'Review ' || i,
96 FLOOR(1 + random() * 10)
97 FROM
98 series.shows s
99 ORDER BY random()
100 LIMIT 1;
101 END LOOP;
102 END $$;
103
104 COMMIT;
```

Data Output Messages Notifications

COMMIT

Query returned successfully in 35 msec.

Выполним актуализацию статистических данных:

ANALYZE VERBOSE;

```
postgres/postgres@PostgreSQL 17* x
postgres/postgres@PostgreSQL 17
Query Query History
1 ANALYZE VERBOSE;
```

Data Output Messages Notifications

```
ИНФОРМАЦИЯ: анализируется "series.seasons"
ИНФОРМАЦИЯ: "seasons": просканировано страниц: 8 из 8, они содержат "живых" строк: 1013, "мёртвых" строк: 0; строк в выборке: 1013, примерное общее число строк: 1013
ИНФОРМАЦИЯ: анализируется "series.shows"
ИНФОРМАЦИЯ: "shows": просканировано страниц: 63 из 63, они содержат "живых" строк: 3005, "мёртвых" строк: 0; строк в выборке: 3005, примерное общее число строк: 3005
ИНФОРМАЦИЯ: анализируется "pg_catalog.pg_type"
ИНФОРМАЦИЯ: "pg_type": просканировано страниц: 15 из 15, они содержат "живых" строк: 631, "мёртвых" строк: 16; строк в выборке: 631, примерное общее число строк: 631
ИНФОРМАЦИЯ: анализируется "series.episodes"
ИНФОРМАЦИЯ: "episodes": просканировано страниц: 8 из 8, они содержат "живых" строк: 1000, "мёртвых" строк: 0; строк в выборке: 1000, примерное общее число строк: 1000
ИНФОРМАЦИЯ: анализируется "series.reviews"
ИНФОРМАЦИЯ: "reviews": просканировано страниц: 9 из 9, они содержат "живых" строк: 1000, "мёртвых" строк: 0; строк в выборке: 1000, примерное общее число строк: 1000
ИНФОРМАЦИЯ: анализируется "people.actors"
ИНФОРМАЦИЯ: "actors": просканировано страниц: 15 из 15, они содержат "живых" строк: 1000, "мёртвых" строк: 0; строк в выборке: 1000, примерное общее число строк: 1000
ИНФОРМАЦИЯ: анализируется "relationships.cast"
ИНФОРМАЦИЯ: "cast": просканировано страниц: 2 из 2, они содержат "живых" строк: 100, "мёртвых" строк: 0; строк в выборке: 100, примерное общее число строк: 100
ИНФОРМАЦИЯ: анализируется "pg_catalog.pg_foreign_table"
ИНФОРМАЦИЯ: "pg_foreign_table": просканировано страниц: 0 из 0, они содержат "живых" строк: 0, "мёртвых" строк: 0; строк в выборке: 0, примерное общее число строк: 0
ИНФОРМАЦИЯ: анализируется "pg_catalog.pg_authid"
ИНФОРМАЦИЯ: "pg_authid": просканировано страниц: 1 из 1, они содержат "живых" строк: 19, "мёртвых" строк: 4; строк в выборке: 19, примерное общее число строк: 19
ИНФОРМАЦИЯ: анализируется "pg_catalog.pg_statistic_ext_data"
ИНФОРМАЦИЯ: "pg_statistic_ext_data": просканировано страниц: 0 из 0, они содержат "живых" строк: 0, "мёртвых" строк: 0; строк в выборке: 0, примерное общее число строк: 0
ИНФОРМАЦИЯ: анализируется "public.accounts"
ИНФОРМАЦИЯ: "accounts": просканировано страниц: 1 из 1, они содержат "живых" строк: 3, "мёртвых" строк: 8; строк в выборке: 3, примерное общее число строк: 3
ИНФОРМАЦИЯ: анализируется "pg_catalog.pg_user_mapping"
ИНФОРМАЦИЯ: "pg_user_mapping": просканировано страниц: 0 из 0, они содержат "живых" строк: 0, "мёртвых" строк: 0; строк в выборке: 0, примерное общее число строк: 0
ИНФОРМАЦИЯ: анализируется "pg_catalog.pg_subscription"
ИНФОРМАЦИЯ: "pg_subscription": просканировано страниц: 0 из 0, они содержат "живых" строк: 0, "мёртвых" строк: 0; строк в выборке: 0, примерное общее число строк: 0
ИНФОРМАЦИЯ: анализируется "pg_catalog.pg_attribute"
ИНФОРМАЦИЯ: "pg_attribute": просканировано страниц: 59 из 59, они содержат "живых" строк: 3348, "мёртвых" строк: 62; строк в выборке: 3348, примерное общее число строк: 3348
ИНФОРМАЦИЯ: анализируется "pg_catalog.pg_proc"
```

Total rows: Query complete 00:00:00.529 CRLF Ln 1, Col 1

Вывод, касаемый таблиц, в которые добавлялись данные:

ИНФОРМАЦИЯ: анализируется "series.seasons"

ИНФОРМАЦИЯ: "seasons": просканировано страниц: 8 из 8, они содержат "живых" строк: 1013, "мёртвых" строк: 0; строк в выборке: 1013, примерное общее число строк: 1013

ИНФОРМАЦИЯ: анализируется "series.shows"

ИНФОРМАЦИЯ: "shows": просканировано страниц: 63 из 63, они содержат "живых" строк: 3005, "мёртвых" строк: 0; строк в выборке: 3005, примерное общее число строк: 3005

ИНФОРМАЦИЯ: анализируется "series.episodes"

ИНФОРМАЦИЯ: "episodes": просканировано страниц: 8 из 8, они содержат "живых" строк: 1000, "мёртвых" строк: 0; строк в выборке: 1000, примерное общее число строк: 1000

ИНФОРМАЦИЯ: анализируется "series.reviews"

ИНФОРМАЦИЯ: "reviews": просканировано страниц: 9 из 9, они содержат "живых" строк: 1000, "мёртвых" строк: 0; строк в выборке: 1000, примерное общее число строк: 1000

ИНФОРМАЦИЯ: анализируется "people.actors"

ИНФОРМАЦИЯ: "actors": просканировано страниц: 15 из 15, они содержат "живых" строк: 1000, "мёртвых" строк: 0; строк в выборке: 1000, примерное общее число строк: 1000

ИНФОРМАЦИЯ: анализируется "relationships.cast"

ИНФОРМАЦИЯ: "cast": просканировано страниц: 2 из 2, они содержат "живых" строк: 100, "мёртвых" строк: 0; строк в выборке: 100, примерное общее число строк: 100

Вывод представляет собой информацию о процессе сканирования и анализа таблиц в базе данных. Каждая строка соответствует отдельной таблице в базе данных, и в ней указаны:

- Название таблицы

- Статус сканирования: указано, сколько страниц таблицы было просканировано
- Количество живых строк
- Количество мертвых строк
- Количество строк в выборке: это количество строк, которое было обработано в текущем запросе или анализе
- Примерное общее число строк: это общее количество строк в таблице, включая как живые, так и мертвые строки

Живые строки — это данные, которые всё ещё актуальны и могут быть использованы для выполнения запросов или операций. Мертвые строки — это данные, которые были удалены, но ещё не освобождены системой

4 Используются ли индексы?

Проверим, используются ли индексы.

The screenshot shows a PostgreSQL query window with the following query: `EXPLAIN SELECT * FROM series.shows WHERE title = 'Game of Thrones';`

The query plan output is as follows:

Step	Operation	Cost	Rows	Width
1	Index Scan using idx_shows_title on shows	(cost=0.28..8.30)	rows=1	width=107
2	Index Cond: (title = 'Game of Thrones':text)			

The screenshot shows a PostgreSQL query window with the following query: `EXPLAIN SELECT * FROM series.shows WHERE year = 2020;`

The query plan output is as follows:

Step	Operation	Cost	Rows	Width
1	Bitmap Heap Scan on shows	(cost=4.47..52.39)	rows=24	width=107
2	Recheck Cond: (year = 2020)			
3	-> Bitmap Index Scan on idx_shows_year	(cost=0.00..4.46)	rows=24	width=0
4	Index Cond: (year = 2020)			

postgres/postgres@PostgreSQL 17* X

postgres/postgres@PostgreSQL 17

No limit

Query Query History

```
4 EXPLAIN SELECT * FROM series.shows WHERE budget > 100000000;
```

Data Output Messages Notifications

Showing rows: 1 to 2

	QUERY PLAN
1	Seq Scan on shows (cost=0.00..100.56 rows=1 width=107)
2	Filter: (budget > '100000000':double precision)

postgres/postgres@PostgreSQL 17* X

postgres/postgres@PostgreSQL 17

No limit

Query Query History

```
7 EXPLAIN SELECT * FROM people.actors WHERE name = 'Emilia Clarke';
```

Data Output Messages Notifications

Showing rows: 1 to 2

	QUERY PLAN
1	Index Scan using idx_actors_name on actors (cost=0.28..8.29 rows=1 width=2...)
2	Index Cond: (name = 'Emilia Clarke':text)

postgres/postgres@PostgreSQL 17* X

postgres/postgres@PostgreSQL 17

No limit

Query Query History

```
10 EXPLAIN SELECT s.*
11 FROM series.shows s
12 JOIN relationships.cast c ON s.id = c.show_id
13 WHERE c.actor_id = (SELECT id FROM people.actors WHERE name = 'Emilia Clarke');
```

Data Output Messages Notifications

Showing rows: 1 to 8

	QUERY PLAN
1	Nested Loop (cost=8.57..19.85 rows=1 width=107)
2	InitPlan 1
3	-> Index Scan using idx_actors_name on actors (cost=0.28..8.29 rows=1 wid...)
4	Index Cond: (name = 'Emilia Clarke':text)
5	-> Seq Scan on "cast" c (cost=0.00..3.25 rows=1 width=4)
6	Filter: (actor_id = (InitPlan 1).col1)
7	-> Index Scan using shows_pkey on shows s (cost=0.28..8.30 rows=1 width=...)
8	Index Cond: (id = c.show_id)

postgres/postgres@PostgreSQL 17* x

postgres/postgres@PostgreSQL 17

Query Query History

16 **EXPLAIN SELECT * FROM series.reviews WHERE show_id = 1;**

Data Output Messages Notifications

Showing rows: 1 to 2

	QUERY PLAN
1	Index Scan using idx_reviews_show_id on reviews (cost=0.28..8.29 rows=1 width=3...
2	Index Cond: (show_id = 1)

postgres/postgres@PostgreSQL 17* x

postgres/postgres@PostgreSQL 17

Query Query History

17 **EXPLAIN SELECT * FROM series.reviews WHERE rating >= 8;**

Data Output Messages Notifications

Showing rows: 1 to 2

	QUERY PLAN
1	Index Scan using idx_reviews_rating on reviews (cost=0.15..4.17 rows=1 width=3...
2	Index Cond: (rating >= 8)

postgres/postgres@PostgreSQL 17* x

postgres/postgres@PostgreSQL 17

Query Query History

20 **EXPLAIN SELECT * FROM series.seasons WHERE show_id = 1;**

Data Output Messages Notifications

Showing rows: 1 to 4

	QUERY PLAN
1	Bitmap Heap Scan on seasons (cost=4.31..11.87 rows=4 width=26)
2	Recheck Cond: (show_id = 1)
3	-> Bitmap Index Scan on idx_seasons_show_id (cost=0.00..4.31 rows=4 widt...
4	Index Cond: (show_id = 1)

The screenshot shows a PostgreSQL query editor interface. The query entered is `EXPLAIN SELECT * FROM series.episodes WHERE season_id = 2;`. The results are displayed in a table with two rows:

	QUERY PLAN
1	Index Scan using idx_episodes_season_id on episodes (cost=0.28..8.29 rows=1 width=2...
2	Index Cond: (season_id = 2)

При анализе с помощью команды EXPLAIN, можно увидеть, что созданные индексы используются для ускорения выполнения данных запросов (запросы будут представлены в скрипте по ссылке из приложения).

5 -----

Данный пункт в ТЗ был помечен, как «дополнительное кармическое задание», так как я не верю в карму, решил его не выполнять 😊

6 -----

Данный пункт в ТЗ был помечен, как «дополнительное кармическое задание», так как я не верю в карму, решил его не выполнять 😊

Заключение

В ходе выполнения работы была разработана система индексов для базы данных, с учетом схемы данных и типичных запросов, характерных для БД сериалов.

Для проверки эффективности предложенных индексов было добавлено значительное количество данных в таблицы, после чего была выполнена команда ANALYZE с ключевым словом VERBOSE, что позволило актуализировать статистику и улучшить выбор планов выполнения запросов. Это обеспечило точность в оценке использования индексов для ускорения выполнения запросов.

С помощью команды EXPLAIN были построены планы выполнения различных запросов, что позволило убедиться, что предложенные индексы действительно используются и позволяют существенно ускорить выполнение типичных операций с данными.

Приложение

- ERD схема;
- скрипт создания индексов;
- скрипт вставки данных;
- скрипт запросов с EXPLAIN;
- отчет (docx);
- отчет (pdf).