

# Индексы: создание и управление

---

**Индексы в базах данных** — это специальные структуры данных, которые позволяют **ускорить поиск** информации в таблицах. Без индексов, для выполнения запроса, СУБД должна просканировать *всю таблицу (full table scan)*, что может быть крайне медленным для больших таблиц. Индексы помогают СУБД находить нужные строки, не просматривая *всю таблицу целиком*.

## Задачи индекса

**Основная задача индекса** — ускорить выполнение операций **SELECT** (выборки данных) в базе данных. Индексы создаются на одном или нескольких столбцах таблицы и содержат упорядоченные данные, позволяющие быстро находить строки, соответствующие условиям запроса.

Когда создается индекс для определенного поля (столбца), СУБД создает отдельную структуру данных (например, B-tree), которая содержит значения этого поля и указатели на соответствующие строки в таблице. Эта структура данных организована таким образом, чтобы обеспечить быстрый поиск нужных значений.

Представьте себе **телефонную книгу**. **Имена** людей — это **индексированное поле**. Благодаря тому, что имена расположены в алфавитном порядке (организованы в индекс), вы можете быстро найти номер телефона нужного человека, не просматривая всю книгу постранично.

## Роли индекса

### Роль индексов в повышении производительности:

- Уменьшение количества читаемых данных:** Индекс позволяет СУБД не просматривать *всю таблицу*, а только небольшую ее часть, содержащую строки, соответствующие условию поиска.
- Ускорение поиска:** Индексы используют структуры данных, оптимизированные для быстрого поиска.
- Оптимизация операций соединения JOIN:** Индексы могут значительно ускорить выполнение операций **JOIN**, особенно если индексы созданы на столбцах, используемых в условиях соединения.
- Ускорение операций сортировки ORDER BY и группировки GROUP BY:** Индексы могут помочь СУБД выполнить сортировку и группировку данных без полного сканирования таблицы.

! Однако, следует помнить, что индексы **ухудшают производительность операций записи** (**INSERT, UPDATE, DELETE**), так как при изменении данных в таблице необходимо также обновлять индексы. Поэтому, необходимо находить баланс между ускорением чтения и замедлением записи.

## Типы индексов

Различные типы индексов используют разные структуры данных, оптимизированные для определенных типов данных и запросов.

## Рассмотрим основные типы:

1. **B-tree (Balanced Tree)**. Дерево сбалансированной высоты, в котором каждый узел содержит ключи и указатели на другие узлы (потомков). Подходит для большинства типов данных и операций сравнения ( $=$ ,  $<$ ,  $>$ ,  $\leq$ ,  $\geq$ , BETWEEN, LIKE). Наиболее распространенный тип индекса.
2. **Hash Index**. Использует хеш-функцию для преобразования ключа в хеш-код, который используется для доступа к данным. Не поддерживает поиск по диапазону или частичному соответствию. В идеальном случае, время поиска не зависит от размера таблицы. Разные ключи могут иметь одинаковые хеш-коды, что может ухудшить производительность. Обычно не рекомендуется для использования в PostgreSQL, так как B-tree часто превосходит его по производительности и универсальности.
3. **GiST**. Универсальная древовидная структура индекса, позволяющая создавать индексы для нестандартных типов данных и операций. Поддерживает различные типы данных (геометрические объекты, текстовые данные и т.д.) и операции (пересечение, вхождение, близость и т.д.). Позволяет определять собственные методы индексирования и поиска. Требует написания специализированных функций для работы с данными.
4. **GIN (Generalized Inverted Index)**. Инвертированный индекс, предназначенный для эффективного поиска по составным значениям (например, массивам, текстовым документам). Индексирует отдельные элементы составного значения (например, слова в текстовом документе). Позволяет быстро находить документы, содержащие определенные слова или элементы. Индекс может быть значительно больше, чем размер таблицы.
5. **SP-GiST**. GiST индекс, оптимизированный для пространственного разделения данных. Эффективен для индексирования географических данных, изображений и других данных, имеющих пространственную структуру. Поддерживает не сбалансированные структуры данных.
6. **BRIN (Block Range Index)**. Индекс, хранящий информацию о диапазонах значений в блоках данных. Предназначен для больших таблиц, в которых физическое расположение данных соответствует порядку значений в столбце. Индекс значительно меньше, чем размер таблицы. Эффективен только если данные физически упорядочены по индексируемому столбцу. Индексирование столбцов с датами, идентификаторами или другими значениями, которые монотонно возрастают или убывают.

## Особенности индекса для операций

**Индексы ускоряют операции чтения, но замедляют операции записи.**

Это связано с тем, что при изменении данных в таблице необходимо также обновлять все индексы, созданные на этой таблице.

**INSERT:** при добавлении новой строки в таблицу необходимо добавить соответствующие записи во все индексы.

**UPDATE:** при изменении значения столбца, входящего в индекс, необходимо обновить записи в индексе.

**DELETE:** при удалении строки из таблицы необходимо удалить соответствующие записи из всех индексов.

 Создавайте индексы только на тех столбцах, которые часто используются в запросах **WHERE**, **JOIN**, **ORDER BY** или **GROUP BY** условиях.

## Выбор типа индекса

**Выбор типа индекса зависит от нескольких факторов:**

1. **Тип данных:** 1.1. Числовые и текстовые данные: B-tree. 1.2. Геопространственные данные: GiST, SP-GiST. 1.3. Составные значения (массивы, текстовые документы): GIN. 1.4. Монотонно возрастающие/убывающие данные: BRIN.
2. **Характер запросов:** 2.1. Точное соответствие (=): B-tree, (Hash - не рекомендуется). 2.2. Сравнение (<, >, <=, >=, BETWEEN, LIKE): B-tree. 2.3. Полнотекстовый поиск: GiST, GIN. 2.4. Пространственные запросы (пересечение, вхождение, близость): GiST, SP-GiST.
3. **Размер таблицы:** 3.1. Большие таблицы с упорядоченными данными: BRIN.

## Мониторинг

**Мониторинг использования индексов:**

1. **Системные представления (pg\_stat\_all\_indexes):** используйте системные представления для получения информации об использовании индексов.
2. **Инструменты мониторинга:** используйте специализированные инструменты мониторинга баз данных для отслеживания производительности запросов и использования индексов.
3. **Логирование запросов:** включите логирование запросов для анализа наиболее частых и медленных запросов.
4. **auto\_explain:** полезное расширение для автоматического протоколирования планов запросов, выполняющихся дольше определенного времени.

## Оптимизация

1. **Удаление неиспользуемых индексов:** удалите индексы, которые не используются или используются очень редко.
2. **Пересоздание индексов:** пересоздайте индексы, которые были повреждены или фрагментированы.
3. **Использование выражений в индексах:** создайте индекс на основе выражения, чтобы оптимизировать запросы, использующие это выражение.
4. **Многоколоночные индексы:** создайте многоколоночный индекс, если часто выполняете запросы, использующие несколько столбцов в условии WHERE.
5. **Фильтрованные индексы:** создайте индекс только для части таблицы, удовлетворяющей определенному условию.
6. **Обновление статистики:** после создания или изменения индексов, выполните команду ANALYZE для обновления статистики о распределении данных в таблице.

## Создание индексов

**Создание БД**

Создайте тестовую базу данных **indexes\_lab** и таблицу для проведения экспериментов.

```

1 ✓ CREATE TABLE products (
2     product_id SERIAL PRIMARY KEY,
3     product_name VARCHAR(255),
4     category VARCHAR(255),
5     price DECIMAL(10, 2),
6     description TEXT,
7     created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT (NOW() AT TIME ZONE 'UTC')
8 );
9
10    -- Заполнение таблицы данными (пример – можно использовать скрипт на Python или другом языке)
11 ✓ INSERT INTO products (product_name, category, price, description)
12     SELECT
13         'Product ' || i,
14         CASE WHEN i % 3 = 0 THEN 'Electronics' WHEN i % 3 = 1 THEN 'Clothing' ELSE 'Home Goods' END,
15         random() * 100,
16         'Description of product ' || i
17     FROM generate_series(1, 100000) AS i;

```

Data Output Сообщения Notifications

INSERT 0 100000

Запрос завершён успешно, время выполнения: 630 msec.

Таблица должна содержать достаточно большое количество данных (минимум 100 000 строк), чтобы разница в производительности запросов с индексами и без индексов была заметна.

Убедимся, что в таблице действительно создалось 100 000 записей, выполнив запрос на подсчет количества строк.

Запрос История запросов	
1	SELECT COUNT(*) FROM products
Data Output Сообщения Notifications	
count	bigint
1	100000

## B-tree

Создайте **B-tree** индекс для столбца product\_name таблицы products.

Запрос История запросов

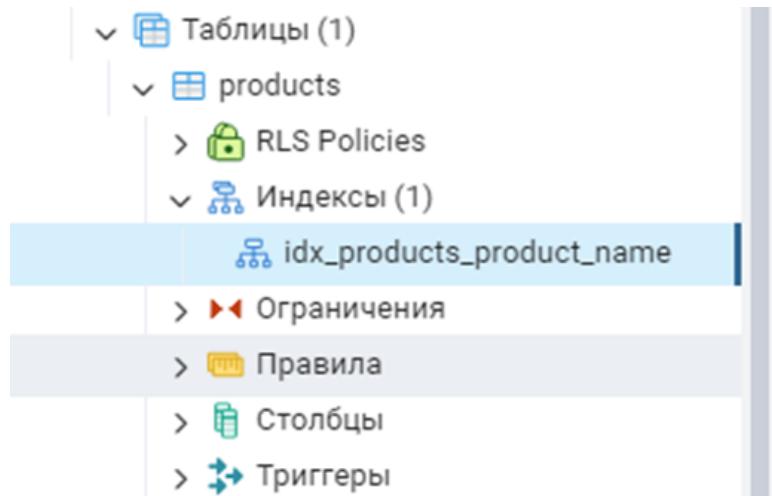
1 CREATE INDEX idx\_products\_product\_name ON products (product\_name);

Data Output Сообщения Notifications

CREATE INDEX

Запрос завершён успешно, время выполнения: 623 msec.

**Убедитесь, что индекс создался.**



## Hash

Создайте Hash индекс для столбца category таблицы products

Запрос История запросов

```
1 CREATE INDEX idx_products_category_hash ON products USING HASH (category);
```

Data Output Сообщения Notifications

CREATE INDEX

Запрос завершён успешно, время выполнения: 511 msec.

**Убедитесь, что индекс создался.**

## GiST

Создайте GiST индекс для столбца description таблицы products

Запрос История запросов

```
1 CREATE EXTENSION pg_trgm;
2 CREATE INDEX idx_products_description_gist ON products USING gist (description gist_trgm_ops); -- Для поиска по частичным соответствиям в текст
```

3

Data Output Сообщения Notifications

CREATE INDEX

Запрос завершён успешно, время выполнения: 1 secs 297 msec.

**Убедитесь, что индекс создался.**

## GIN

Создайте GIN индекс для столбца description таблицы products

Запрос История запросов

```
1 CREATE INDEX idx_products_description_gin ON products USING gin (description gin_trgm_ops);
```

Data Output Сообщения Notifications

```
CREATE INDEX
```

Запрос завершён успешно, время выполнения: 555 msec.

**Убедитесь, что индекс создался.**

## SP-GiST

Создание SP-GiST индекса для текстового столбца description **не имеет** особого смысла.

**Пример (для демонстрации синтаксиса, но не применимо к текстовому полю):** допустим у нас есть столбец location типа point.

```
CREATE INDEX idx_products_location_spgist ON products USING spgist (location);
```

## BRIN

Создайте BRIN (Block Range Index) индекс для столбца created\_at таблицы products

Запрос История запросов

```
1 CREATE INDEX idx_products_created_at_brin ON products USING brin (created_at);  
2
```

Data Output Сообщения Notifications

```
CREATE INDEX
```

Запрос завершён успешно, время выполнения: 39 msec.

**Убедитесь, что индекс создался.**

В итоге должно получиться 5 индексов в таблице

Таблицы (1)  
products  
RLS Policies  
Индексы (5)  
idx\_products\_category\_hash  
idx\_products\_created\_at\_btree  
idx\_products\_description\_gin  
idx\_products\_description\_gist  
idx\_products\_product\_name

Выполните несколько запросов к таблице products, не использующих столбцы, по которым созданы индексы. Запишите время выполнения каждого запроса.

Запрос по первичному ключу (будет быстрым даже без индекса).

Запрос История запросов

```
1 EXPLAIN ANALYZE SELECT * FROM products WHERE product_id = 50000;
2
3
```

Data Output Сообщения Graph Visualiser × Notifications

QUERY PLAN  
text

1	Index Scan using products_pkey on products (cost=0.29..8.31 rows=1 width=69) (actual time=0.012..0.013 rows=1 loop...
2	Index Cond: (product_id = 50000)
3	Planning Time: 0.107 ms
4	Execution Time: 0.055 ms

Запрос по столбцу без индекса (изначально).

```

1 EXPLAIN ANALYZE SELECT * FROM products WHERE price > 75;
2
3

```

Data Output Сообщения Graph Visualiser X Notifications

<b>QUERY PLAN</b>								
text								
1	Seq Scan on products (cost=0.00..2543.00 rows=24625 width=69) (actual time=0.013..9.318 rows=24927 loops=1)							
2	Filter: (price > '75'::numeric)							
3	Rows Removed by Filter: 75073							
4	Planning Time: 0.095 ms							
5	Execution Time: 9.816 ms							

План запроса показывает, что для выполнения запроса использовался последовательный просмотр (полный скан) всей таблицы "products", что является не самым эффективным способом. Отсутствие индекса на столбце "price" приводит к тому, что база данных вынуждена просмотреть все строки, чтобы отфильтровать нужные.

## Анализ

- Seq Scan on products (Последовательный просмотр таблицы "products"):** 1.1. База данных просматривает каждую строку в таблице "products". 1.2. cost=0.00..2543.00: Ориентировочная стоимость этой операции (в абстрактных единицах) — от 0 до 2543. Чем выше стоимость, тем больше ресурсов требуется. 1.3. rows=24625: Оценка количества строк, которые вернет эта операция (первоначальная оценка количества строк в таблице) - 24625. 1.4. width=69: Примерная ширина каждой строки (в байтах) - 69 байт. 1.5. (actual time=0.013..9.318): Фактическое время выполнения этой операции - от 0.013 миллисекунд до 9.318 миллисекунд. 1.6. rows=24927: Фактическое количество строк, которые были прочитаны при просмотре таблицы - 24927.
- Filter: (price > '75'::numeric) (Фильтр: (цена > 75 (как число))):** 2.1. После просмотра каждой строки, строки отфильтровываются. В данном случае, остаются только те, у которых значение в столбце "price" больше 75.
- Rows Removed by Filter: 75073 (Строк удалено фильтром: 75073):** 3.1. Фильтр удалил 75073 строки. Это означает, что из общего количества строк таблицы, 75073 строки имели значение "price" меньше или равное 75.
- Planning Time: 0.095 ms (Время планирования: 0.095 миллисекунд):** 4.1. Базе данных потребовалось 0.095 миллисекунд для того, чтобы построить план выполнения запроса.
- Execution Time: 9.816 ms (Время выполнения: 9.816 миллисекунд):** 5.1. Общее время выполнения всего запроса составило 9.816 миллисекунд. Выполните аналогичные запросы,

использующие столбцы, по которым созданы индексы (рисунки 54 – 57). Запишите время выполнения каждого запроса.

The screenshot shows the pgAdmin interface with the following details:

- Top Bar:** Shows "Запрос История запросов" (Query History) and a query editor containing:

```
1 EXPLAIN ANALYZE SELECT * FROM products WHERE product_name = 'Product 50000'; -- B-tree индекс
```
- Navigation Bar:** Includes tabs for "Data Output", "Сообщения" (Messages), "Graph Visualiser", and "Notifications".
- Toolbar:** Features icons for new connection, file operations (New, Open, Save, Print, Copy, Paste, Delete, Import, Export, Refresh, Help), and a search/filter icon.
- Table View:** A results table with the following rows:

	QUERY PLAN	
1	Index Scan using idx_products_product_name on products (cost=0.42..8.44 rows=1 width=69) (actual time=0.086..0.086 rows=1 loop... Index Cond: ((product_name)::text = 'Product 50000'::text)	
2	Planning Time: 0.154 ms	
3	Execution Time: 0.099 ms	

## Анализ запроса по индексу B-tree

Запрос История запросов	
1	EXPLAIN ANALYZE SELECT * FROM products WHERE category = 'Electronics'; -- Hash индекс
Data Output	Сообщения Graph Visualiser X Notifications
QUERY PLAN	
text	
1	Seq Scan on products (cost=0.00..2543.00 rows=33197 width=69) (actual time=0.011..6.980 rows=33333 loops=)
2	Filter: ((category)::text = 'Electronics'::text)
3	Rows Removed by Filter: 66667
4	Planning Time: 0.093 ms
5	Execution Time: 7.547 ms

## Анализ запроса по индексу Hash

Запрос История запросов	
1 EXPLAIN ANALYZE SELECT * FROM products WHERE description LIKE '%product%'; -- GIST/GIN индекс	
Data Output Сообщения Graph Visualiser × Notifications	
QUERY PLAN	
text	
1	Seq Scan on products (cost=0.00..2543.00 rows=99990 width=69) (actual time=0.007..11.399 rows=100000 loops=)
2	Filter: (description ~~ '%product%':text)
3	Planning Time: 0.178 ms
4	Execution Time: 13.101 ms

## Анализ запроса по индексу Gist/GIN

Запрос История запросов

```
1 EXPLAIN ANALYZE SELECT * FROM products WHERE created_at BETWEEN '2023-01-01' AND '2024-01-01'; -- BRIN индекс
```

Data Output Сообщения Graph Visualiser X Notifications

QUERY PLAN  
text

1	Bitmap Heap Scan on products (cost=12.03..1441.40 rows=1 width=69) (actual time=0.322..0.322 rows=0 loops=1)
2	Recheck Cond: ((created_at >= '2023-01-01 00:00:00'::timestamp without time zone) AND (created_at <= '2024-01-01 00:00:00'::timestamp without time zo...)
3	-> Bitmap Index Scan on idx_products_created_at_brin (cost=0.00..12.03 rows=9091 width=0) (actual time=0.320..0.320 rows=0 loops=1)
4	Index Cond: ((created_at >= '2023-01-01 00:00:00'::timestamp without time zone) AND (created_at <= '2024-01-01 00:00:00'::timestamp without time z...)
5	Planning Time: 0.135 ms
6	Execution Time: 0.533 ms

## Анализ запроса по индексу BRIN

**Сравните время выполнения запросов с использованием и без использования индексов.**  
**Оцените влияние каждого типа индекса на производительность запросов.**

Используйте системное представление pg\_stat\_all\_indexes для получения информации об использовании индексов. Это представление содержит статистику о количестве чтений индекса и таблицы.

**Проанализируйте статистику использования индексов. Обратите внимание на индексы, которые используются редко или не используются вообще. Если индекс занимает много места, но не используется, рассмотрите возможность его удаления, но пока что не удаляйте.**

Запрос История запросов

```
1 v
      SELECT
        schemaname,
        relname,
        indexrelname,
        idx_scan,
        pg_size.pretty(pg_relation_size(indexrelid)) AS index_size
      FROM pg_stat_all_indexes
      WHERE schemaname = 'public' -- Замените на схему вашей таблицы, если необходимо
      ORDER BY idx_scan DESC;
```

Data Output Сообщения Graph Visualiser X Notifications

Закрыть

	schemaname name	relname name	indexrelname name	idx_scan bigint	index_size text
1	public	products	products_pkey	2	2208 kB
2	public	products	idx_products_product_name	1	3104 kB
3	public	products	idx_products_created_at_brin	1	24 kB
4	public	products	idx_products_category_hash	0	6056 kB
5	public	products	idx_products_description_gist	0	14 MB
6	public	products	idx_products_description_gin	0	4352 kB

## Оптимизация индексов

Самостоятельно оптимизируйте существующие индексы (обратите внимание, что ниже представлены только примеры).

**Необходимо на основе Вашего анализа выполнить оптимизацию:**

- Удалите индексы, которые не используются или используются очень редко** (на основе анализа статистики из pg\_stat\_all\_indexes). DROP INDEX idx\_products\_category\_hash; -- Пример удаления Hash индекса.
- Пересоздайте индексы, которые были повреждены или фрагментированы.** Это может улучшить производительность запросов. REINDEX INDEX idx\_products\_product\_name; --Пример пересоздания индекса.
- Создайте индекс на основе выражения,** чтобы оптимизировать запросы, использующие это выражение. Например, для поиска товаров по имени в нижнем регистре.

Запрос История запросов

```
1 CREATE INDEX idx_products_product_name_lower ON products (lower(product_name));
2 EXPLAIN ANALYZE SELECT * FROM products WHERE lower(product_name) = 'product 50000';
```

Data Output Сообщения Graph Visualiser × Notifications

The screenshot shows the PostgreSQL Explain Analyze interface. At the top, there are tabs for 'Запрос' (Query), 'История запросов' (Query History), 'Data Output' (selected), 'Сообщения' (Messages), 'Graph Visualiser', and 'Notifications'. Below the tabs is a toolbar with icons for copy, paste, clear, download, and refresh. The main area is titled 'QUERY PLAN' and contains the following text:

```
text
1 Bitmap Heap Scan on products (cost=12.29..980.24 rows=500 width=69) (actual time=0.411..0.413 rows=1 loops=1)
2 Recheck Cond: (lower((product_name)::text) = 'product 50000'::text)
3 Heap Blocks: exact=1
4 -> Bitmap Index Scan on idx_products_product_name_lower (cost=0.00..12.17 rows=500 width=0) (actual time=0.403..0.404 rows=1 loop...
5 Index Cond: (lower((product_name)::text) = 'product 50000'::text)
6 Planning Time: 3.805 ms
7 Execution Time: 0.446 ms
```

- Создайте многоколоночный индекс,** если часто выполняете запросы, использующие несколько столбцов в условии WHERE. Порядок столбцов в многоколоночном индексе важен.

Запрос История запросов

```
1 CREATE INDEX idx_products_category_price ON products (category, price);
2 EXPLAIN ANALYZE SELECT * FROM products WHERE category = 'Electronics' AND price > 50;
```

Data Output Сообщения Graph Visualiser × Notifications

The screenshot shows the PostgreSQL Explain Analyze interface. At the top, there are tabs for 'Запрос' (Query), 'История запросов' (Query History), 'Data Output' (selected), 'Сообщения' (Messages), 'Graph Visualiser', and 'Notifications'. Below the tabs is a toolbar with icons for copy, paste, clear, download, and refresh. The main area is titled 'QUERY PLAN' and contains the following text:

```
text
1 Bitmap Heap Scan on products (cost=472.39..2011.21 rows=16388 width=69) (actual time=3.071..5.349 rows=16528 loops=1)
2 Recheck Cond: (((category)::text = 'Electronics'::text) AND (price > '50'::numeric))
3 Heap Blocks: exact=1293
4 -> Bitmap Index Scan on idx_products_category_price (cost=0.00..468.30 rows=16388 width=0) (actual time=2.950..2.951 rows=16528 loop...
5 Index Cond: (((category)::text = 'Electronics'::text) AND (price > '50'::numeric))
6 Planning Time: 4.120 ms
7 Execution Time: 5.703 ms
```

**5. Создайте индекс только для части таблицы, удовлетворяющей определенному условию.**

Это может быть полезно, если запросы часто фильтруются по определенному значению.

Запрос История запросов

```
1 CREATE INDEX idx_products_expensive ON products (product_name) WHERE price > 75;
2 EXPLAIN ANALYZE SELECT * FROM products WHERE product_name = 'Product 123' AND price > 75;
3
```

Data Output Сообщения Graph Visualiser X Notifications

QUERY PLAN text

1	Index Scan using idx_products_expensive on products (cost=0.29..8.30 rows=1 width=69) (actual time=0.033..0.034 rows=1 loop...
2	Index Cond: ((product_name)::text = 'Product 123'::text)
3	Planning Time: 2.833 ms
4	Execution Time: 0.046 ms

**6. После создания или изменения индексов, выполните команду ANALYZE для обновления статистики о распределении данных в таблице. Это позволит оптимизатору запросов принимать более эффективные решения.**

Запрос История запросов

```
1 ANALYZE products;
```

Data Output Сообщения Graph Visualiser X Notifications

ANALYZE

Запрос завершён успешно, время выполнения: 579 msec.

**Самостоятельное задание: оптимизируйте работу базы данных «Склад» с помощью создания индексов и различных манипуляций с ними. Проведите анализ производительности.**

**! После выполнения самостоятельного задания, удалите базу данных indexes\_lab.**