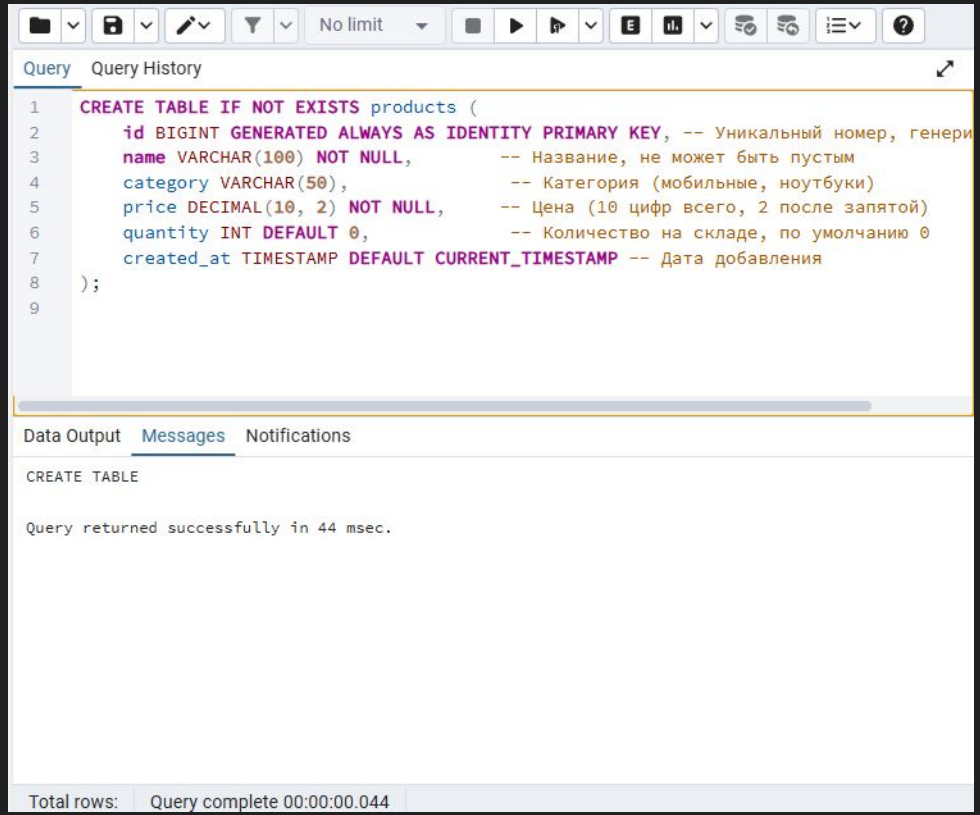
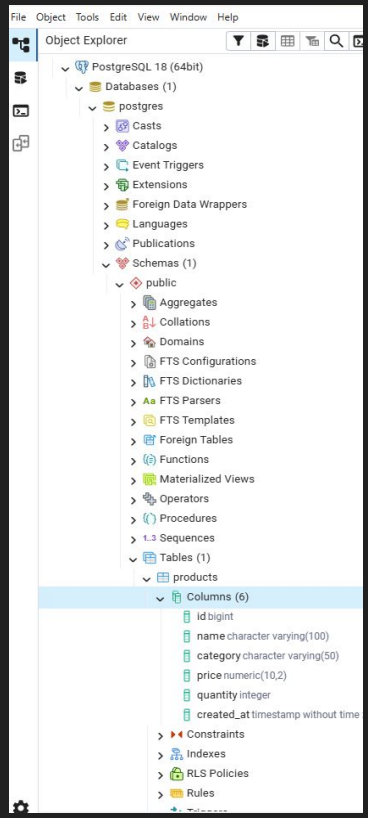
SQL на практике: создаем таблицу товаров и взаимодействуем с ней. Часть 1

Грачев Д.Г.

Создание таблицы продуктов

```
-- создаем таблицу продуктов если не существует

CREATE TABLE IF NOT EXISTS products (

    id BIGINT GENERATED ALWAYS AS IDENTITY PRIMARY KEY, -- Уникальный номер,
генерируется сам

    name VARCHAR(100) NOT NULL,        -- Название, не может быть пустым

    category VARCHAR(50),               -- Категория (мобильные, ноутбуки)

    price DECIMAL(10, 2) NOT NULL,     -- Цена (10 цифр всего, 2 после запятой)

    quantity INT DEFAULT 0,             -- Количество на складе, по умолчанию 0

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP -- Дата добавления

  );
```

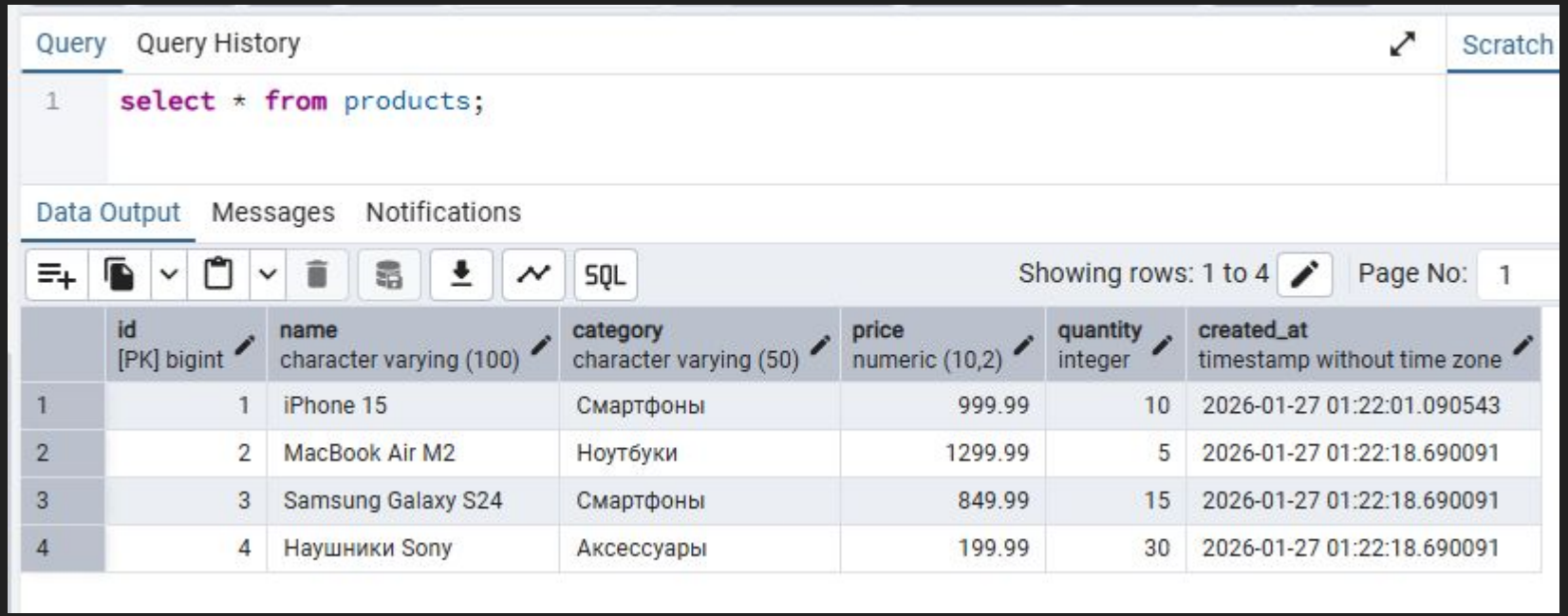

No limit
Query
Query History
1 CREATE TABLE IF NOT EXISTS products (
2 id BIGINT GENERATED ALWAYS AS IDENTITY PRIMARY KEY, -- Уникальный номер, генери
3 name VARCHAR(100) NOT NULL, -- Название, не может быть пустым
4 category VARCHAR(50), -- Категория (мобильные, ноутбуки)
5 price DECIMAL(10, 2) NOT NULL, -- Цена (10 цифр всего, 2 после запятой)
6 quantity INT DEFAULT 0, -- Количество на складе, по умолчанию 0
7 created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP -- Дата добавления
8);
9
Data Output
Messages
Notifications
CREATE TABLE
Query returned successfully in 44 msec.
Total rows:
Query complete 00:00:00.044


File Object Tools Edit View Window Help
Object Explorer
PostgreSQL 18 (64bit)
Databases (1)
postgres
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas (1)
public
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (1)
products
Columns (6)
id bigint
name character varying(100)
category character varying(50)
price numeric(10,2)
quantity integer
created_at timestamp without time
Constraints
Indexes
RLS Policies
Rules

Добавление товара

```sql
-- Добавляем один товар
INSERT INTO products (name, category, price, quantity)
VALUES ('iPhone 15', 'Смартфоны', 999.99, 10);


-- Добавляем несколько товаров за раз
INSERT INTO products (name, category, price, quantity) VALUES
('MacBook Air M2', 'Ноутбуки', 1299.99, 5),
('Samsung Galaxy S24', 'Смартфоны', 849.99, 15),
  ('Наушники Sony', 'Аксессуары', 199.99, 30);
```

Проверка результата



Query | Query History | Scratch

```
1  select * from products;
```

Data Output | Messages | Notifications

Showing rows: 1 to 4 | Page No: 1

	id [PK] bigint	name character varying (100)	category character varying (50)	price numeric (10,2)	quantity integer	created_at timestamp without time zone
1	1	iPhone 15	Смартфоны	999.99	10	2026-01-27 01:22:01.090543
2	2	MacBook Air M2	Ноутбуки	1299.99	5	2026-01-27 01:22:18.690091
3	3	Samsung Galaxy S24	Смартфоны	849.99	15	2026-01-27 01:22:18.690091
4	4	Наушники Sony	Аксессуары	199.99	30	2026-01-27 01:22:18.690091

В чем подвох?

1. Если в таблице миллионы записей, то ваша система ляжет
2. Нужны ли все колонки ?
3. Как проходит запрос на самом деле?

Пример CRUD операций

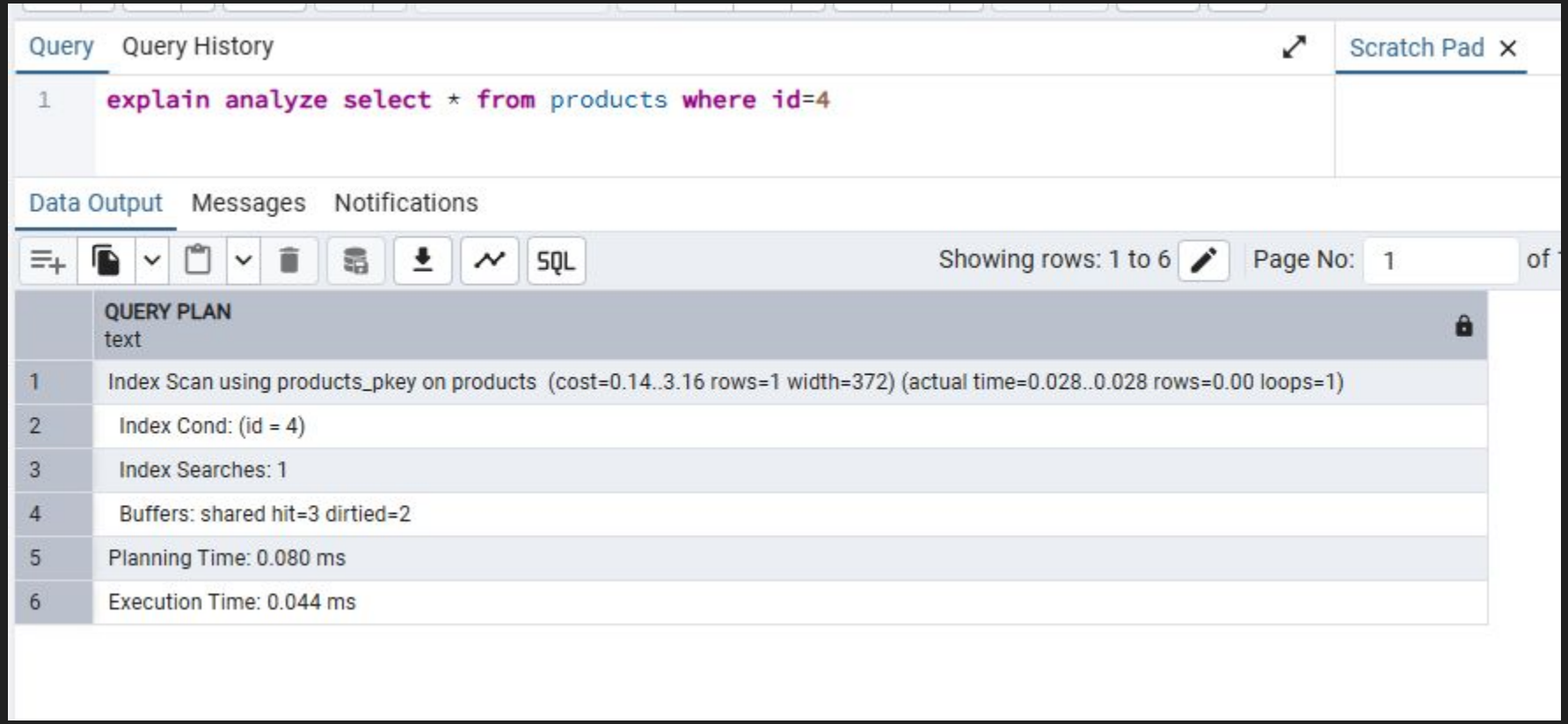
1. CRUD - create/read/update/delete
2. create рассмотрели на втором слайде
3. select * from products limit 10 - первые 10 записей
4. update products set name='Наушники Panasonic' where id=4
5. delete from products where id=4

explain analyze

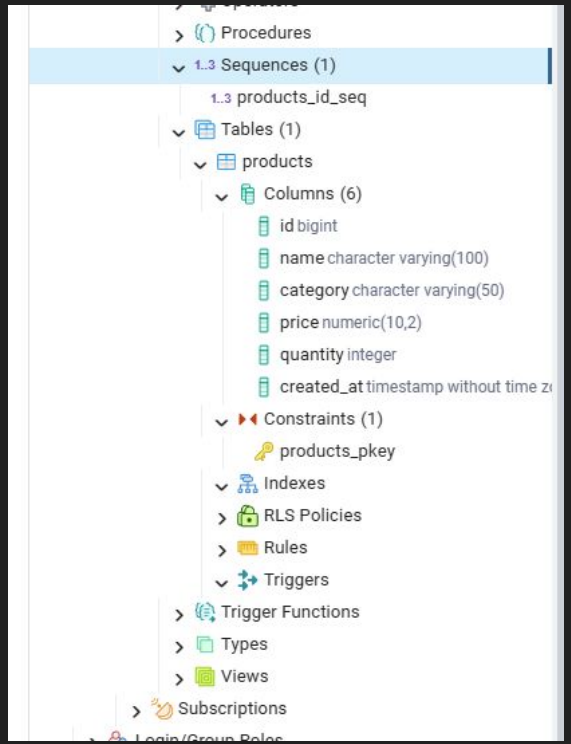
EXPLAIN ANALYZE — это мощная SQL-команда (поддерживаемая в PostgreSQL, MySQL 8.0+, MariaDB и др.), которая используется для профилирования и оптимизации запросов.

- **EXPLAIN:** показывает только **план выполнения** — «теоретический» маршрут, который планировщик базы данных собирается использовать на основе статистики (оценки стоимости и количества строк).
- **EXPLAIN ANALYZE:** фактически **выполняет запрос** и дополняет план **реальными данными** о времени работы (actual time) и количестве обработанных строк (rows) на каждом этапе

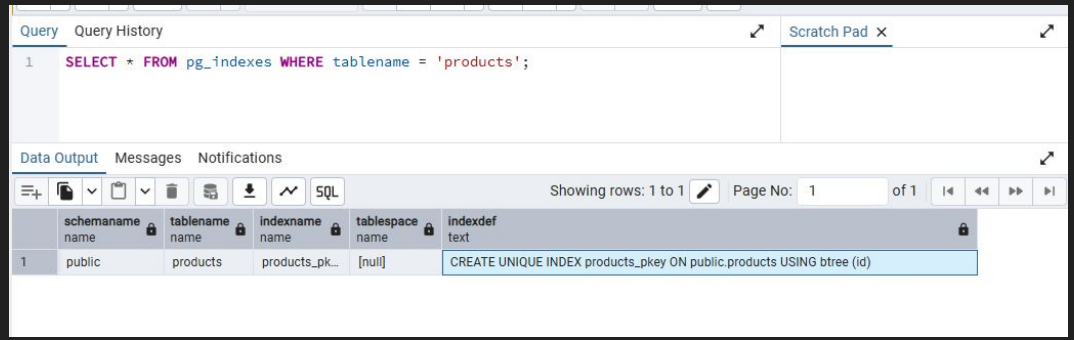
Пример использования explain analyze



Неявное создание индексов



SELECT * FROM pg_indexes WHERE tablename = 'products';



Изменение колонок в существующих таблицах

Так как время не стоит на месте, система должна иметь возможность быть гибкой для внесения изменений даже если первоначально она спроектирована идеально. На самом деле только если она хорошо спроектирована она позволит безболезненно внести изменения.

Далее рассмотрим прямые и непрямые способы изменения таблиц и их результатов.

Изменение колонок в существующих таблицах

`ALTER TABLE table_name ADD COLUMN column_name data_type [constraints]` - Базовый синтаксис для добавления колонки в существующую таблицу

`ALTER TABLE table_name ALTER COLUMN column_name [action]` - изменение типа данных колонки

`ALTER TABLE table_name DROP COLUMN column_name [CASCADE | RESTRICT]` - удаление колонки

`CREATE [OR REPLACE] VIEW view_name AS SELECT column1, column2, … FROM table_name WHERE condition` - создание представления для отображения без фактического изменения самих таблиц

Примеры

```sql
— создадим таблицу сотрудников
CREATE TABLE employees (
    id BIGSERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    salary DECIMAL(10, 2)
  );
```

```sql
— Добавим колонку почтового ящика
ALTER TABLE employees
ADD COLUMN email VARCHAR(100)
  NOT NULL DEFAULT 'no-email@company.com';

-- Меняем VARCHAR(50) на VARCHAR(100)
ALTER TABLE employees
  ALTER COLUMN first_name TYPE VARCHAR(100);

 -- Разрешаем NULL
ALTER TABLE employees
  ALTER COLUMN salary DROP NOT NULL;

  -- Переименование колонки
  ALTER TABLE employees
    RENAME COLUMN first_name TO given_name;
```

Примеры

```
-- Удаляем неиспользуемую колонку
ALTER TABLE employees DROP COLUMN metadata;

-- Удаляем колонку, даже если на нее ссылаются
ALTER TABLE employees DROP COLUMN email CASCADE;

-- Не удалить, если есть зависимости (по умолчанию)
ALTER TABLE employees DROP COLUMN salary RESTRICT;

-- Не упадет с ошибкой, если колонки нет
ALTER TABLE employees DROP COLUMN IF EXISTS bonus;
```

View

VIEW — это сохраненный SQL-запрос, который ведет себя как таблица.

```sql
-- Создание представления
CREATE VIEW hr_employee_view AS
SELECT
    id,
    first_name || ' ' || last_name AS full_name,
    email,
    salary,
    hire_date,
    CASE
        WHEN salary > 100000 THEN 'Senior'
        WHEN salary > 50000 THEN 'Middle'
        ELSE 'Junior'
    END AS level
FROM employees
  WHERE is_active = TRUE;
```

```sql
-- Используем как обычную таблицу

SELECT * FROM hr_employee_view
WHERE level = 'Senior'
  ORDER BY salary DESC;
```

Явное создание индексов

Допустим нам необходимо ускорить поиск по имени продукта, тогда добавляем на него индекс:

CREATE INDEX IF NOT EXISTS product_name_idx ON products(name);

ПС неявно он создает именно BTree индекс, но можно указать явно:

CREATE INDEX IF NOT EXISTS product_name_idx ON products USING BTREE (name);

Но стоит учитывать что обычный B-Tree индекс чувствителен к регистру. Для регистронезависимого поиска можно:

```
CREATE INDEX IF NOT EXISTS product_name_lower_idx ON products(LOWER(name));
```

Но стоит помнить что для полнотекстного поиска лучше использовать дополнительные инструменты.

Набор ключевых слов SQL на каждый день

```
SELECT, FROM, WHERE, INSERT, UPDATE, DELETE

JOIN, ORDER BY, GROUP BY, HAVING

CREATE, ALTER, DROP

PRIMARY KEY, FOREIGN KEY, UNIQUE

BEGIN, COMMIT, ROLLBACK

EXPLAIN, ANALYZE
```

Немного не правда ли?
Но на самом деле ключевые слова зависят от поддержки конкретной бд. К примеру в Postgresql их немного больше 300!

Расширенный набор ключевых слов

ADD ALL ALTER ANALYZE AND ANY ARRAY_AGG AS ASC AT AUTHORIZATION BEGIN BETWEEN BIGINT BIGSERIAL BOOLEAN BY CASCADE CASE CAST CHAR CHECK CLUSTER COLUMN COMMIT CONSTRAINT COPY COUNT CREATE CROSS DATABASE DATE DECIMAL DEFAULT DELETE DESC DISTINCT DO DOUBLE DROP ELSE END EXCEPT EXISTS EXPLAIN FALSE FOREIGN FROM FULL FUNCTION GENERATED GRANT GROUP HAVING IDENTITY ILIKE IN INDEX INNER INSERT INT INTEGER INTERSECT INTERVAL INTO IS JOIN JSON JSONB KEY LAG LEAD LEFT LIKE LIMIT MAX MIN MONEY NATURAL NOT NULL NUMERIC OFFSET ON ONLY OR ORDER OUTER OVER PARTITION PRECISION PRIMARY RANK REAL REFERENCES REINDEX REVOKE RIGHT ROLLBACK ROW_NUMBER SAVEPOINT SCHEMA SELECT SEQUENCE SERIAL SET SIMILAR SMALLINT SOME SUM TABLE TEXT THEN TIME TIMESTAMP TO TRANSACTION TRUE TRUNCATE UNION UNIQUE UPDATE USING UUID VALUES VARCHAR VACUUM VIEW WHEN WHERE WITH

Самые важные:

CREATE TABLE SELECT INSERT UPDATE DELETE FROM WHERE ORDER BY GROUP BY PRIMARY KEY FOREIGN KEY UNIQUE NOT NULL AND OR

Средний уровень:

JOIN INNER LEFT RIGHT FULL CROSS COUNT SUM AVG MIN MAX DISTINCT LIMIT OFFSET HAVING CHECK DEFAULT ALTER DROP TRUNCATE BEGIN COMMIT ROLLBACK

Продвинутый уровень:

EXPLAIN ANALYZE WITH RECURSIVE GENERATED ALWAYS AS IDENTITY JSONB ILIKE SIMILAR TO OVER PARTITION BY LAG LEAD ROW_NUMBER RANK DENSE_RANK AT TIME ZONE COPY VACUUM REINDEX

Практические задания

1. Добавьте в таблицу продуктов новые 3-4 записи
2. Проанализируйте с помощью explain / explain analyze запросы на вставку. В чем разница?
3. Создайте новую таблицу категорий продуктов
4. Свяжите таблицу продуктов и категорий в отношении многие ко многим
5. Добавьте в обе таблицы данные и свяжите их
6. Проанализируйте как работает теперь select во время join запроса к товару с категориями. Можно ли улучшить?

Ресурсы

- https://postgrespro.ru/windows - установщик Postgresql для винды
- https://www.postgresql.org/docs/current/index.html - официальная актуальная документация Postgresql
- https://sqliteonline.com/ - песочница sql в различным базах данных
- https://www.pgadmin.org/download/ - PgAdmin
- Основы технологий баз данных (Новиков Б. А., Графеева Н. Г., Горшкова Е. А)
- https://dbeaver.io - универсальный бесплатный инструмент для управления базами данных и SQL-клиент