

PostgreSQL 11 Administration CookBook

Chapter-1: First Steps

Upcode Software
Engineer Team

-2024-



CONTENT

1. **Introducing PostgreSQL 11**
2. **OmniDB**
3. **Practice**
4. **References**



CONTENT

1. Представляем **PostgreSQL 11**
2. **OmniDB**
3. Практика
4. Ссылки

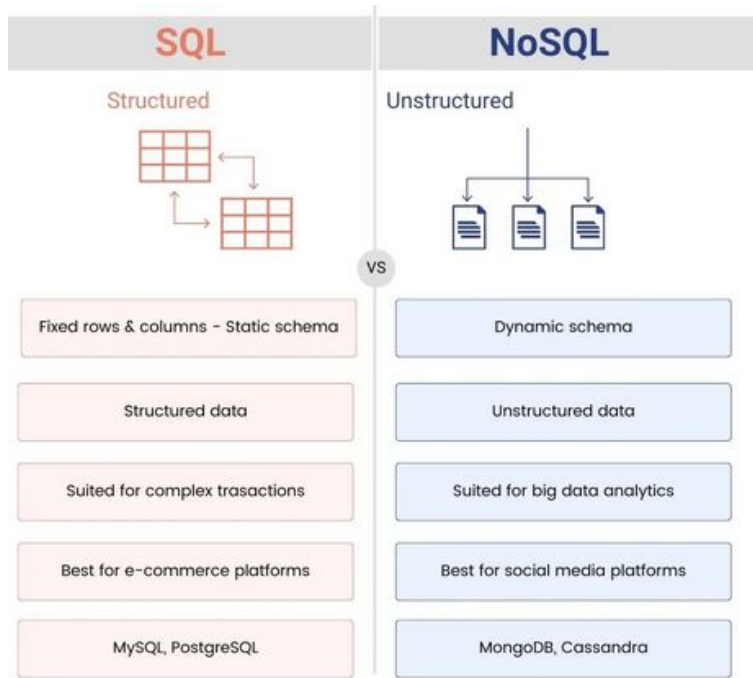


Introducing PostgreSQL 11

PostgreSQL обладает следующими основными особенностями:

- Превосходное соответствие стандартам SQL, вплоть до SQL: 2016;
- Клиент-серверная архитектура;
- Имеет высокую параллельную структуру, в которой читатели и записи не блокируют друг друга;
- Легко конфигурируется и расширяется для многих типов приложений;
- Обладает отличной масштабируемостью и производительностью, с обширными возможностями настройки;
- Предлагает поддержку многих видов моделей данных, такие как реляционные, постреляционные (массивы, вложенные отношения через типы записей), документные (JSON и XML) и ключ/значение.

Отличие SQL и NoSQL





Introducing PostgreSQL 11

Отличие PostgreSQL от других(1/2)

- Надежность(Robustness)
- Безопасность (Security)
- Простота использования (Ease of use)
- Растяжимость (Extensibility)
- Производительность и параллелизм (Performance and concurrency)
- Масштабируемость (Scalability)
- SQL and NoSQL data models (SQL and NoSQL data models)



Introducing PostgreSQL 11

Отличие PostgreSQL от других(2/2)

- Популярность (Popularity)
- Коммерческая поддержка (Commercial support)
- Финансирование научно-исследовательских и опытно-конструкторских работ (Research and development funding)



Introducing PostgreSQL 11

- **Надежность(Robustness):**
 - Репликация баз данных поддерживается по умолчанию. Синхронная репликация может обеспечить доступность и защиту данных на уровне выше 5 девяток (99,999%) при правильной настройке и управлении или даже выше при соответствующей избыточности
- **Безопасность (Security):**
 - Доступ к PostgreSQL контролируется с помощью правил доступа на основе хоста. Аутентификация является гибкой и подключаемой, что позволяет легко интегрироваться с любой внешней архитектурой безопасности. Новейший механизм аутентификации Salted Challenge Response Authentication Mechanism (SCRAM) обеспечивает полную 256-битную защиту.



Introducing PostgreSQL 11

- **Простота использования (Ease of use):**
 - Текстовые данные поддерживаются одним типом данных, который позволяет хранить от 1 байта до 1 гигабайта. Это хранилище оптимизировано несколькими способами, поэтому 1 байт хранится эффективно, а гораздо большие значения автоматически управляются и сжимаются.
- **Растяжимость (Extensibility):**
 - PostgreSQL поддерживает пользовательские типы данных, операторы, индексы, функции и языки.
 - Для PostgreSQL доступно множество расширений, в том числе расширение PostGIS, которое предоставляет возможности географической информационной системы (ГИС) мирового класса.



Introducing PostgreSQL 11

- **Производительность и параллелизм (Performance and concurrency):**
 - PostgreSQL 11 может достигать значительно более **1 000 000** операций чтения в секунду на 4-сокетном сервере и обеспечивает более **30 000** транзакций записи в секунду с полной надежностью
 - PostgreSQL предоставляет MVCC, который позволяет читателям и писателям избегать блокировки друг друга.
- **Масштабируемость (Scalability):**
 - PostgreSQL 11 хорошо масштабируется на одном узле до четырёх процессорных сокетов. PostgreSQL эффективно поддерживает до сотен активных сессий и до тысяч подключенных сессий при использовании пула сеансов. Дальнейшая масштабируемость достигается в каждом ежегодном выпуске



Introducing PostgreSQL 11

- **SQL and NoSQL data models (SQL and NoSQL data models):**
 - Документоцентричная база данных также возможна с использованием текстовых, XML- и двоичных типов данных JSON (JSONB) PostgreSQL, поддерживаемых индексами, оптимизированными для документов, и возможностями полнотекстового поиска.



Introducing PostgreSQL 11

- **Популярность (Popularity):**
 - Различные опросы показали, что PostgreSQL является излюбленной базой данных для создания новых приложений корпоративного класса.
 - Набор функций PostgreSQL привлекает серьезных пользователей, у которых есть серьезные приложения. Компании, предоставляющие финансовые услуги, могут быть самой большой группой пользователей PostgreSQL, хотя правительства, телекоммуникационные компании и многие другие сегменты также являются сильными пользователями.
 - Эта популярность распространяется по всему миру; Япония, Эквадор, Аргентина и Россия имеют очень большие группы пользователей, как и США, Европа и Австралия.



Introducing PostgreSQL 11

- **Коммерческая поддержка (Commercial support):**
 - Авторы (Джанни и Саймон) работают во 2-м квадранте, который обеспечивает коммерческую поддержку PostgreSQL с открытым исходным кодом, предлагая поддержку 24/7 на английском и испанском языках со временем исправления ошибок.
- **Финансирование научно-исследовательских и опытно-конструкторских работ (Research and development funding):**
 - PostgreSQL был первоначально разработан как исследовательский проект в Калифорнийском университете в Беркли в конце 1980-х и начале 1990-х годов. Дальнейшая работа велась добровольцами до конца 1990-х годов.



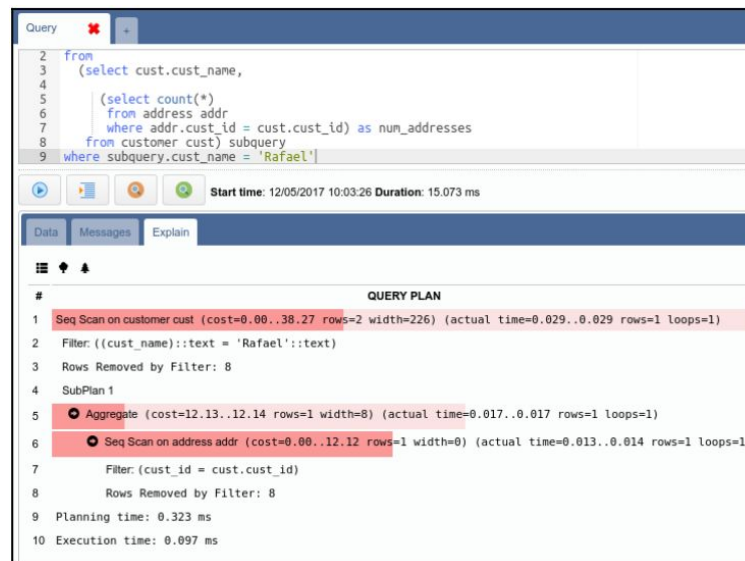
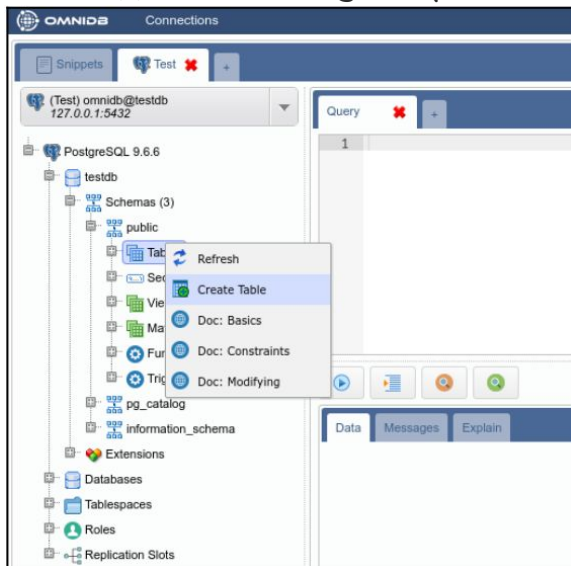
Connecting to the PostgreSQL server

Проверка подключились ли в нужном месте и правильным образом:

- `select current_database();` - выводит название текущего базы данных;
- `select current_user;` - выводит имя текущего пользователя;
- `select inet_server_addr(), inet_server_port();` - показывает IP-адрес и порт текущего соединения;
- `select version();` - показывает текущую версию PostgreSQL.
- `\conninfo` - выводит информацию про соединение

OmniDB

- OmniDB предназначена для доступа к PostgreSQL, MySQL, MariaDB и Oracle в одном интерфейсе, хотя она обеспечивает полный набор функций для базы данных PostgreSQL



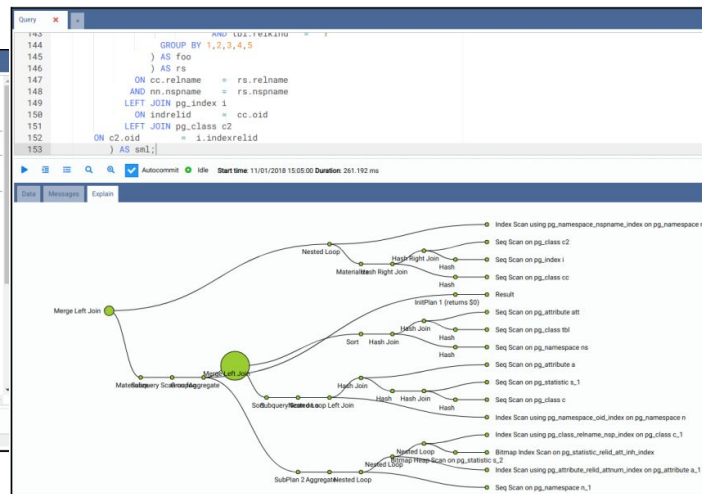


OmniDB

- Или, если вы предпочитаете командную строку, попробуйте вкладку **Console**:

```
>> Console tab. Type the commands in the editor below this box. \? to view command list.
>> \dt
SELECT 0
>> \d+ categories
+-----+
| Column | Type          | Modifiers          | Storage | Stats target | Description |
+-----+
| category | integer       | not null default nextval('categories_category_seq'::regclass) | plain   | None         | None        |
| categoryname | character varying(50) | not null          | extended | None         | None        |
+-----+
Indexes:
    "categories_pkey" PRIMARY KEY, btree (category)
Has OIDs: no

>> \timing
Timing is on.
>> select *
from categories
+-----+
| category | categoryname |
+-----+
| 1        | Action       |
| 2        | Animation    |
| 3        | Children     |
| 4        | Classics     |
| 5        | Comedy       |
+-----+
```



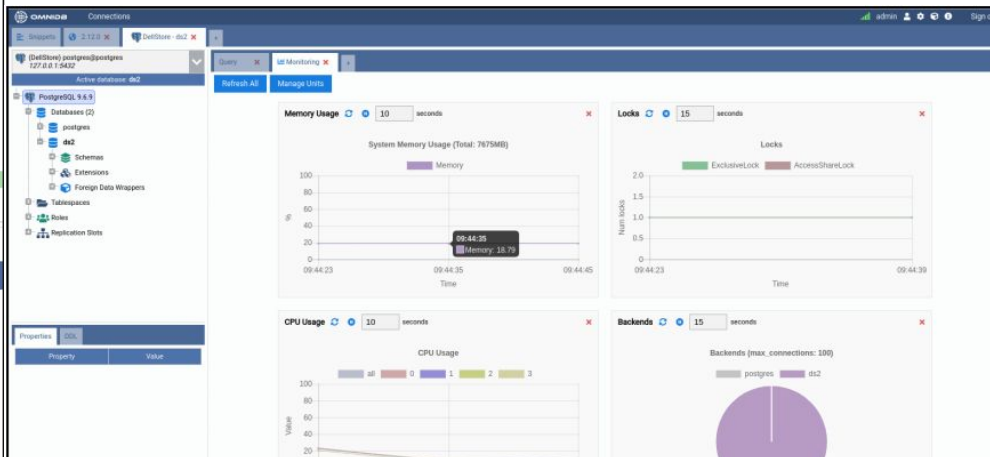
OmniDB

New Connection														
		Groups:		Select group ▼		New Group								
Group	Technology	Server	Port	Database	User	Title	SSH Tunnel	SSH Server	SSH Port	SSH User	SSH Password	SSH Key	Actions	
<input type="checkbox"/>	postgresql	127.0.0.1	5432	testdb	testuser	PostgreSQL	<input type="checkbox"/>		22				<input type="checkbox"/> <input checked="" type="checkbox"/>	
<input type="checkbox"/>	oracle	127.0.0.1	1521	XE	SYSTEM	Oracle	<input type="checkbox"/>		22				<input type="checkbox"/> <input checked="" type="checkbox"/>	
<input type="checkbox"/>	mysql	127.0.0.1	3306	employees	root	MySQL	<input type="checkbox"/>		22				<input type="checkbox"/> <input checked="" type="checkbox"/>	
<input type="checkbox"/>	postgresql	127.0.0.1	5432	remotedb	remoteuser	Remote PostgreSQL	<input checked="" type="checkbox"/>	example.org	22	admin	*****		<input type="checkbox"/> <input checked="" type="checkbox"/>	

Query ☒ Create Function ☒ Query ☒ Debugger: fnc_count_vowels ☒

```
8 BEGIN
9   str := upper(p_input);
10  ret := 0;
11  i := 1;
12  len := length(p_input);
13  WHILE i <= len LOOP
14    IF substr(str, i, 1) in ('A', 'E', 'I', 'O', 'U') THEN
15      SELECT pg_sleep(1) INTO tmp;
16      ret := ret + 1;
17    END IF;
18    i := i + 1;
19  END LOOP;
20  RETURN ret;
21 END;
```

	Variable	Attribute	Type	Value
1	\$1		text	The quick brown fox jumps o...
2	found		bool	t
3	str		text	THE QUICK BROWN FOX JU...
4	ret		int4	11
5	i		int4	45
6	len		int4	44
7	tmp		text	





Практика

```
select current_time;
select version();
select date_trunc('second', current_timestamp - pg_postmaster_start_time()) as
uptime;
select current_timestamp - pg_postmaster_start_time();
select date_trunc('second', current_timestamp - pg_postmaster_start_time()) as
uptime;

select pg_listening_channels();

select pg_database_size(current_database());

select sum(pg_database_size());
```

Практика - Создание таблицы

```
/*CREATE TABLE: cars */  
create table cars(  
    brand VARCHAR(255),  
    model VARCHAR(255),  
    year INT  
);
```

- Отображение таблицы:

```
SELECT * FROM  
cars;
```

Практика - Вставка

- Вставка данные в таблицу:

```
/*INSERT DATA TO TABLE: cars */  
INSERT INTO cars (brand, model, year)  
VALUES ('Ford', 'Mustang', 1964);
```

- Результат:

	<input type="checkbox"/> brand <input type="text"/>	<input type="checkbox"/> model <input type="text"/>	<input type="checkbox"/> year <input type="text"/>
1	Ford	Mustang	1964
2	Volvo	p1800	1968
3	BMW	M1	1978
4	Toyota	Celica	1975

- Вставка несколько строк:

```
/*INSERT MULTIPLE ROWS TO TABLE:  
cars*/  
INSERT INTO cars (brand, model,  
year)  
VALUES  
('Volvo', 'p1800', 1968),  
('BMW', 'M1', 1978),  
('Toyota', 'Celica', 1975);
```

Практика - Выбор данных

- Указав имена столбцов, мы можем выбрать, какие столбцы следует выбрать:

```
select brand, year from cars;
```



	brand	year
1	Ford	1964
2	Volvo	1968
3	BMW	1978
4	Toyota	1975

- Возвращение всех столбцов:

```
select * from cars;
```



	brand	model	year
1	Ford	Mustang	1964
2	Volvo	p1800	1968
3	BMW	M1	1978
4	Toyota	Celica	1975

Практика - Изменение таблицы(1/2)

- Чтобы добавить столбец в существующую таблицу, мы должны использовать **ALTER TABLE**.

```
/*ADD COLUMN NAMED color*/  
ALTER TABLE cars  
add color varchar(255);
```



	brand	model	year	color
1	Ford	Mustang	1964	<null>
2	Volvo	p1800	1968	<null>
3	BMW	M1	1978	<null>
4	Toyota	Celica	1975	<null>



Практика - Изменение таблицы(2/2)

- **ALTER TABLE** используется для добавления, удаления или изменения столбцов в существующей таблице.
- **ALTER TABLE** также используется для добавления и удаления различных ограничений в существующей таблице.

Практика - Изменение значение строки

- Инструкция **UPDATE** используется для изменения значений в существующих записях таблицы

```
UPDATE cars  
SET color = 'red'  
WHERE brand = 'Volvo';
```



	brand	model	year	color
1	Ford	Mustang	1964	<null>
2	BMW	M1	1978	<null>
3	Toyota	Celica	1975	<null>
4	Volvo	p1800	1968	red

Практика - Изменение столбца(1/2)

- Чтобы изменить столбец, используется **ALTER COLUMN** и ключевое слово **TYPE**, за которым следует новый тип данных:

```
/*CHANGE year COLUMN from INT to  
VARCHAR(4)*/  
ALTER TABLE cars  
    ALTER COLUMN year TYPE  
    VARCHAR(4);
```



	brand	model	year	color
1	Ford	Mustang	1964	<null>
2	BMW	M1	1978	<null>
3	Toyota	Celica	1975	<null>
4	Volvo	p1800	1968	red

Практика - Изменение столбца(2/2)

- **Примечание:** Некоторые типы данных не могут быть преобразованы, если в столбце есть значение. Например, числа всегда можно преобразовать в текст, но текст не всегда можно преобразовать в числа.

```
ALTER TABLE cars ALTER COLUMN color TYPE  
VARCHAR (30);
```



	brand	model	year	color
1	Ford	Mustang	1964	<null>
2	BMW	M1	1978	<null>
3	Toyota	Celica	1975	<null>
4	Volvo	p1800	1968	red

Практика - Удаление столбца

- Чтобы удалить столбец, используйте команду **DROP COLUMN** (УДАЛИТЬ СТОЛБЕЦ):

```
/*DELETING COLUMN: color*/  
ALTER TABLE cars DROP COLUMN color;
```



	brand	model	year
1	Ford	Mustang	1964
2	BMW	M1	1978
3	Toyota	Celica	1975
4	Volvo	p1800	1968

Практика - Удаление строки

- **DELETE** используется для удаления существующих записей в таблице.
- **Примечание:** Если вы забудите предложение **WHERE**, все записи в таблице будут удалены!

```
/*DELETE ALL RECORDS WHERE  
  BRAND IS 'VOLVO'*/  
DELETE FROM cars  
WHERE brand = 'Volvo';
```



	brand	model	year
1	Ford	Mustang	1964
2	BMW	M1	1978
3	Toyota	Celica	1975
4	Volvo	p1800	1968

	brand	model	year
1	Ford	Mustang	1964
2	BMW	M1	1978
3	Toyota	Celica	1975



Практика - Удаление таблицы

- **DROP TABLE** используется для удаления существующей таблицы в базе данных.
- **Примечание:** Будьте осторожны, прежде чем удалять таблицу. Удаление таблицы приведет к потере всей информации, хранящейся в таблице!

```
/*DELETING TABLE: cars*/  
DROP TABLE cars;
```



```
[42P01] ERROR: relation "cars" does not exist  
Позиция: 15
```



Практика - Создание демо базы данных

- Мы создадим эти 6 таблиц в нашей базе данных PostgreSQL:
 - **categories**
 - **customers**
 - **products**
 - **orders**
 - **order_details**
 - **testproducts**

Практика - Создание демо базы данных

- CATEGORIES:

```
/*CREATE TABLE: categories*/  
CREATE TABLE categories(  category_id SERIAL NOT NULL PRIMARY KEY ,  
                           category_name VARCHAR(255), description VARCHAR(255));
```



```
/*INSERT DATA INTO TABLE: categories*/  
INSERT INTO categories (category_name, description)  
VALUES  
    ('Beverages', 'Soft drinks, coffees, teas, beers, and ales' ),  
    ('Condiments', 'Sweet and savory sauces, relishes, spreads, and seasonings' ),  
    ('Confections', 'Desserts, candies, and sweet breads' ),  
    ('Dairy Products', 'Cheeses'),  
    ('Grains/Cereals', 'Breads, crackers, pasta, and cereal' ),  
    ('Meat/Poultry', 'Prepared meats'),  
    ('Produce', 'Dried fruit and bean curd' ),  
    ('Seafood', 'Seaweed and fish');
```

Практика - Создание демо базы данных

- CUSTOMERS:

```
/*CREATE TABLE: customers*/  
CREATE TABLE customers (  
    customer_id SERIAL NOT NULL PRIMARY KEY ,  
    customer_name VARCHAR(255),  
    contact_name VARCHAR(255),  
    address VARCHAR(255),  
    city VARCHAR(255),  
    postal_code VARCHAR(255),  
    country VARCHAR(255)  
);
```



```
/*INSERT DATA TO TABLE: customers*/  
INSERT INTO customers (customer_name, contact_name, address, city, postal_code,  
country)  
VALUES  
    ('Alfreds Futterkiste', 'Maria Anders', 'Obere Str. 57', 'Berlin', '12209',  
'Germany'));
```


Практика - Создание демо базы данных

- PRODUCTS:

```
/*CREATE TABLE: products*/  
CREATE TABLE products (  
    product_id SERIAL NOT NULL PRIMARY KEY ,  
    product_name VARCHAR(255),  
    category_id INT,  
    unit VARCHAR(255),  
    price DECIMAL(10, 2)  
);
```



```
/*INSERT INTO DATA TO TABLE: products */  
INSERT INTO products (product_id, product_name, category_id, unit, price)  
VALUES  
    (1, 'Chais', 1, '10 boxes x 20 bags', 18);
```

Практика - Создание демо базы данных

- **ORDERS:**

```
/*CREATE TABLE: orders*/  
CREATE TABLE orders (  
    order_id SERIAL NOT NULL PRIMARY KEY ,  
    customer_id INT,  
    order_date DATE  
);
```



```
/*INSERT DATA INTO TABLE: orders*/  
INSERT INTO orders (order_id, customer_id, order_date)  
VALUES  
    (10248, 90, '2021-07-04'),
```

Практика - Создание демо базы данных

- ORDER_DETAILS:

```
/*CREATE TABLE: order_details*/  
CREATE TABLE order_details (  
    order_detail_id SERIAL NOT NULL PRIMARY KEY ,  
    order_id INT,  
    product_id INT,  
    quantity INT  
);
```



```
/*INSERT DATA INTO TABLE: order_details*/  
INSERT INTO order_details (order_id, product_id, quantity)  
VALUES  
    (10248, 11, 12);
```

Практика - Создание демо базы данных

- TESTPRODUCTS:

```
/*CREATE TABLE: testproducts*/  
CREATE TABLE testproducts (  
    testproduct_id SERIAL NOT NULL PRIMARY KEY ,  
    product_name VARCHAR(255),  
    category_id INT  
);
```










```
/*INSERT DATA TO TABLE: testproducts*/  
INSERT INTO testproducts (product_name, category_id)  
VALUES  
    ('Johns Fruit Cake' , 3);
```



Практика - Синтаксис(1/2)

- Операторы в предложении **WHERE**:

	равняется
	Меньше, чем
	Больше, чем
	Меньше или равно
	Больше или равно
	Не равно
	Не равно

Практика - Синтаксис(2/2)

LIKE	Проверьте, соответствует ли значение шаблону (с учетом регистра).
ILIKE	Проверьте, соответствует ли значение шаблону (без учета регистра).
AND	Логичный И
OR	Логический ИЛИ
IN	Проверьте, находится ли значение в диапазоне значений
BETWEEN	Проверьте, находится ли значение в диапазоне значений
IS NULL	Проверьте, является ли значение нулевым
NOT	Приводит к отрицательному результату, например, НЕ НРАВИТСЯ, НЕ В, НЕ МЕЖДУ



Практика - Синтаксис пример

- Оператор **=** используется, когда вы хотите вернуть все записи, в которых столбец равен указанному значению:

```
SELECT * FROM cars WHERE brand =  
      'Volvo';
```

- Оператор **<** используется, когда вы хотите вернуть все записи, в которых значение столбца меньше указанного значения.

```
SELECT * FROM cars WHERE year < 1975;
```

- Оператор **>** используется, когда вы хотите вернуть все записи, в которых значение столбца превышает указанное значение.

```
SELECT * FROM cars WHERE year > 1975;
```



Практика - Синтаксис пример

- Оператор `<=` используется, когда вы хотите вернуть все записи, в которых столбец меньше или равен указанному значению.

```
SELECT * FROM cars WHERE year <= 1975;
```

- Оператор `>=` используется, когда вы хотите вернуть все записи, в которых значение столбца больше или равно указанному значению.

```
SELECT * FROM cars WHERE year >= 1975;
```

- Оператор `<>` используется, когда вы хотите вернуть все записи, в которых столбец не равен указанному значению:

```
SELECT * FROM cars WHERE brand <> 'Volvo';
```




Практика - Синтаксис пример

- Верните все записи, в которых бренд НЕ указан как "Volvo":

```
SELECT * FROM cars WHERE brand != 'Volvo';
```

- Оператор LIKE используется, когда вы хотите вернуть все записи, в которых столбец соответствует заданному шаблону:
 - Шаблоном может быть абсолютное значение, например "Volvo", или подстановочный знак, имеющий специальное значение.
 - В сочетании с оператором LIKE часто используются два подстановочных знака:
 - Знак процента % обозначает ноль, один или несколько символов.
 - Знак подчеркивания _ обозначает один символ.

```
SELECT * FROM cars WHERE model LIKE 'M%';
```

Практика - Синтаксис пример

- Верните все записи, в которых бренд НЕ указан как "Volvo":

```
SELECT * FROM cars WHERE brand != 'Volvo';
```

- Оператор LIKE используется, когда вы хотите вернуть все записи, в которых столбец соответствует заданному шаблону:
 - Шаблоном может быть абсолютное значение, например "Volvo", или подстановочный знак, имеющий специальное значение.
 - В сочетании с оператором **LIKE** часто используются два подстановочных знака:
 - Знак процента % обозначает ноль, один или несколько символов.
 - Знак подчеркивания _ обозначает один символ.

```
SELECT * FROM cars WHERE model LIKE 'M%';
```

- **Примечание:** Оператор **LIKE** чувствителен к регистру символов.

Практика - Синтаксис пример

- То же, что и оператор **LIKE**, но **ILIKE** не чувствителен к регистру.

```
SELECT * FROM cars WHERE model ILIKE 'm%';
```

- Логический оператор **AND** используется, когда вы хотите проверить несколько условий:

```
SELECT * FROM cars WHERE brand = 'Volvo' AND year = 1968;
```

- The logical **OR** operator is used when you can accept that only one of many conditions is true:

```
SELECT * FROM cars WHERE brand = 'Volvo' OR year = 1975;
```

Практика - Синтаксис пример

- Оператор **IN** используется, когда значение столбца совпадает с любым из значений в списке:

```
SELECT * FROM cars WHERE brand IN ('Volvo', 'Mercedes', 'Ford');
```

- Оператор **BETWEEN** используется для проверки того, находится ли значение столбца в пределах заданного диапазона значений:

```
SELECT * FROM cars WHERE year BETWEEN 1970 AND 1980;
```

- Примечание:** Оператор BETWEEN включает значения **from** и **to**, что означает, что в приведенном выше примере результат будет включать также автомобили, выпущенные в 1970 и 1980 годах.

Практика - Синтаксис пример

- Оператор **IS NULL** используется для проверки того, является ли значение столбца нулевым:

```
SELECT * FROM cars WHERE model IS  
NULL;
```

Оператор NOT

- Оператор **NOT** может использоваться вместе с операторами **LIKE**, **ILIKE**, **IN**, **BETWEEN** и **NULL** для изменения истинности оператора.

```
SELECT * FROM cars WHERE brand NOT LIKE 'B%';
```

- Вернется все записи, в которых название бренда начинается не с буквы "b" (без учета регистра):

```
SELECT * FROM cars WHERE brand NOT ILIKE 'b%';
```



Практика - Синтаксис пример

- Верните все записи, в которых марка отсутствует в этом списке: ("Volvo", "Mercedes", "Ford"):

```
SELECT * FROM cars WHERE brand NOT IN ('Volvo', 'Mercedes',  
                                         'Ford');
```

- Верните все записи, в которых год НЕ находится между 1970 и 1980 годами:

```
SELECT * FROM cars WHERE year NOT BETWEEN 1970 AND 1980;
```

- **Примечание:** Оператор "NOT BETWEEN" исключает значения "от" и "до", что означает, что в приведенном выше примере результат не будет включать автомобили, выпущенные в 1970 и 1980 годах.



Практика - Синтаксис пример

- Возвращает все записи, в которых значение model НЕ равно null:

```
SELECT * FROM cars WHERE model IS NOT  
NULL;
```

- **Примечание:** В таблице cars нет столбцов с нулевыми значениями, поэтому в приведенном выше примере будут возвращены все 4 строки.



Практика - Выбор данные

- Чтобы извлечь данные из базы данных, мы используем **SELECT**.

```
SELECT customer_name, country FROM  
customers;
```

- Укажите ***** вместо имен столбцов, чтобы выбрать все столбцы:

```
SELECT * FROM customers;
```




Практика - Выбор данные

- Оператор **SELECT DISTINCT** используется для возврата только различных значений.
- Внутри таблицы столбец часто содержит много повторяющихся значений, и иногда вы хотите перечислить только разные (distinct) значения.

```
SELECT DISTINCT country FROM  
customers;
```

- Мы также можем использовать ключевое слово **DISTINCT** в сочетании с оператором **COUNT**, который в приведенном ниже примере вернет количество различных стран в таблице customers.

```
SELECT COUNT(DISTINCT country) FROM  
customers;
```



Практика - Фильтр данных

- Предложение **WHERE** используется для фильтрации записей.
- Оно используется для извлечения только тех записей, которые удовлетворяют указанному условию.
- Если мы хотим вернуть только те записи, в которых городом является Лондон, мы можем указать это в предложении **WHERE**:

```
SELECT * FROM customers WHERE city = 'London';
```

- PostgreSQL требует заключать текстовые значения в кавычках
- Однако числовые поля не следует заключать в кавычки:

```
SELECT * FROM customers WHERE customer_id =  
19;
```

Примечание: Кавычки вокруг числовых полей не приведут к сбою, но рекомендуется всегда записывать числовые значения без кавычек.



Практика - Фильтр данных

- Используйте оператор `>`, чтобы вернуть все записи, в которых значение `customer_id` превышает 80:

```
SELECT * FROM customers WHERE customer_id > 80;
```

Примечание: Кавычки вокруг числовых полей не приведут к сбою, но рекомендуется всегда записывать числовые значения без кавычек.



Практика - Сортировка данных

- Ключевое слово **ORDER BY** используется для сортировки результатов по возрастанию или убыванию.

```
SELECT * FROM products ORDER BY price;
```

- Ключевое слово **ORDER BY** по умолчанию сортирует записи по возрастанию. Чтобы отсортировать записи по убыванию, используйте ключевое слово **DESC**.

```
SELECT * FROM products ORDER BY price  
DESC;
```



Практика - Сортировка данных

- Для строковых значений ключевое слово **ORDER BY** будет располагаться в алфавитном порядке:

```
SELECT * FROM products ORDER BY product_name;
```

- Чтобы отсортировать таблицу в обратном алфавитном порядке, используйте ключевое слово **DESC**:

```
SELECT * FROM products ORDER BY product_name DESC;
```



Практика - Лимит

- Предложение **LIMIT** используется для ограничения максимального количества возвращаемых записей.

```
SELECT * FROM customers LIMIT 20;
```

- Предложение **OFFSET** используется для указания, с чего начать выбор записей для возврата.
- Если вы хотите вернуть 20 записей, но начать с номера 40, вы можете использовать как **LIMIT**, так и **OFFSET**.

```
SELECT * FROM customers LIMIT 20 OFFSET  
40;
```

Примечание: Первая запись имеет номер 0, поэтому, когда вы указываете **OFFSET 40**, это означает начало с записи под номером 41.



Практика - Нахождение мин и макс

- Функция **MIN()** возвращает наименьшее значение выбранного столбца.

```
SELECT MIN(price) FROM products;
```

- Функция **MAX()** возвращает наибольшее значение выбранного столбца.

```
SELECT MAX(price) FROM products;
```

- При использовании функций **MIN()** или **MAX()** возвращаемый столбец по умолчанию будет называться min или max. Чтобы присвоить столбцу новое имя, используйте ключевое слово **AS**.

```
SELECT MIN(price) AS lowest_price FROM products;
```



Практика - Нахождение количества

- Функция `COUNT()` возвращает количество строк, соответствующих указанному критерию.
- Если указанным критерием является имя столбца, функция `COUNT()` возвращает количество столбцов с таким именем.

Примечание: НУЛЕВЫЕ значения не учитываются.

```
SELECT COUNT(customer_id) FROM customers;
```

- Указав предложение `WHERE`, вы можете, например, вернуть количество клиентов, прибывших из Лондона:

```
SELECT COUNT(customer_id) FROM customers WHERE city = 'London';
```




Практика - Нахождение суммы

- Функция **SUM()** возвращает общую сумму числового столбца.
- Следующая инструкция SQL находит сумму количественных полей в таблице **order_details**:

```
SELECT SUM(quantity) FROM order_details;
```

Примечание: НУЛЕВЫЕ значения игнорируются.

Практика - Нахождение среднего значения

- Функция **AVG()** возвращает среднее значение числового столбца.

```
SELECT AVG(price) FROM products;
```

Примечание: НУЛЕВЫЕ значения игнорируются.

- В приведенном выше примере была возвращена средняя цена всех продуктов, результат был равен **28,86636363636364**.
- Мы можем использовать оператор **::NUMERIC** для округления средней цены до числа с двумя знаками после запятой:

```
SELECT AVG(price)::NUMERIC(10,2) FROM products;
```

Практика - Оператор LIKE

- Оператор **LIKE** используется в предложении **WHERE** для поиска заданного шаблона в столбце.
- В сочетании с оператором **LIKE** часто используются два подстановочных знака:
 - **%** Знак процента обозначает ноль, один или несколько символов
 - **_** Знак подчеркивания обозначает один единственный символ

```
SELECT * FROM customers WHERE customer_name LIKE 'A%';
```

- Чтобы вернуть записи, содержащие определенную букву или фразу, добавьте **%** до и после буквы или фразы.

```
SELECT * FROM customers WHERE customer_name LIKE '%A%';
```

Практика - Оператор ILIKE и знак “_”

Примечание: Оператор **LIKE** чувствителен к регистру, если вы хотите выполнить поиск без учета регистра, используйте вместо него оператор **ILIKE**.

```
SELECT * FROM customers WHERE customer_name ILIKE '%A%';
```

- Чтобы вернуть записи, заканчивающиеся определенной буквой или фразой, добавьте % перед буквой или фразой.

```
SELECT * FROM customers WHERE customer_name LIKE  
'%en';
```

- Знак **_** обозначает один символ.
- Это может быть любой символ или число, но каждый символ **_** обозначает один и только один символ.

```
SELECT * FROM customers WHERE city LIKE 'L_nd__';
```

Практика - Операторы NOT и IN

- Оператор **IN** позволяет указать список возможных значений в предложении **WHERE**.
- Оператор **IN** является сокращением для нескольких условий **OR**.

```
SELECT * FROM customers WHERE country IN ('Germany', 'France',  
                                           'UK');
```

Используя ключевое слово **NOT** перед оператором **IN**, вы возвращаете все записи, которые не являются ни одним из значений в списке.

```
SELECT * FROM customers WHERE country NOT IN ('Germany', 'France',  
                                              'UK');
```

Вы также можете использовать инструкцию **SELECT** внутри круглых скобок, чтобы вернуть все записи, которые находятся в результате выполнения инструкции **SELECT**.

```
SELECT * FROM customers WHERE customer_id IN (SELECT customer_id FROM  
                                              orders);
```

Практика - Операторы NOT и IN

- Результат в приведенном выше примере вернул 89 записей, это означает, что есть 2 клиента, которые не размещали никаких заказов.
- Давайте проверим, правильно ли это, используя оператор NOT IN

```
SELECT * FROM customers WHERE customer_id NOT IN (SELECT customer_id FROM orders);
```



	customer_id	customer_name	contact_name	address	city	postal_code
1	22	FISSA Fabrica Inter. Salchichas S.A.	Diego Roel	C/ Morazarzal, 86	Madrid	28034
2	57	Paris specialites	Marie Bertrand	265, boulevard Charonne	Paris	75012



Практика - Оператор BETWEEN

- Оператор **BETWEEN** выбирает значения в пределах заданного диапазона. Значения могут быть числами, текстом или датами.
- Оператор **BETWEEN** является включающим: включаются начальные и конечные значения.

```
SELECT * FROM Products WHERE Price BETWEEN 10 AND 15;
```

- The **BETWEEN** operator can also be used on text values.
- The result returns all records that are *alphabetically* between the specified values.

```
SELECT * FROM Products WHERE product_name BETWEEN 'Pavlova' AND  
                                'Tofu';
```



Практика - Оператор ORDER BY

- Если мы добавим к приведенному выше примеру предложение **ORDER BY**, его будет немного легче читать:

```
SELECT * FROM Products WHERE product_name BETWEEN 'Pavlova' AND 'Tofu' ORDER BY  
product_name;
```

- Оператор **BETWEEN** также может использоваться для значений даты.

```
SELECT * FROM orders WHERE order_date BETWEEN '2023-04-12' AND '2023-05-05';
```




Практика - Оператор AS

- Псевдонимы SQL используются для присвоения таблице или столбцу в таблице временного имени.
- Псевдонимы часто используются для того, чтобы сделать имена столбцов более удобочитаемыми.
- Псевдоним существует только на время выполнения запроса.
- Псевдоним создается с помощью ключевого слова AS.

```
SELECT customer_id AS id FROM  
customers;
```

- На самом деле, вы можете пропустить ключевое слово **AS** и получить тот же результат:

```
SELECT customer_id id FROM customers;
```

Практика - Оператор AS

- Ключевое слово **AS** часто используется, когда два или более поля объединяются в одно.
- Для объединения двух полей используйте **||**.

```
SELECT product_name || unit AS product FROM products;
```

Примечание: В результате приведенного выше примера мы пропустили пробел между **product_name** и **unit**. Чтобы добавить пробел при объединении, используйте **|| ' ' ||**.

```
SELECT product_name || ' ' || unit AS product FROM  
products;
```

Если вы хотите, чтобы ваш псевдоним содержал один или несколько пробелов, например "Мои замечательные продукты", заключите его в двойные кавычки.

```
SELECT product_name AS "My Great Products" FROM  
products;
```

Практика - Оператор JOIN

- Предложение **JOIN** используется для объединения строк из двух или более таблиц на основе связанного столбца между ними.
- Давайте рассмотрим выборку из таблицы products:

product_id	product_name	category_id
33	Geitost	4
34	Sasquatch Ale	1
35	Steeleye Stout	1
36	Inlagd Sill	8

Затем просмотрите подборку из таблицы категорий:

category_id	category_name
1	Beverages
2	Condiments
3	Confections
4	Dairy Products

Практика - Оператор JOIN

- Обратите внимание, что столбец `category_id` в таблице `products` относится к столбцу `category_id` в таблице `categories`. Связь между двумя таблицами, приведенными выше, заключается в столбце `category_id`.
- Затем мы можем создать следующую инструкцию SQL (с объединением), которая выбирает записи, имеющие совпадающие значения в обеих таблицах:

```
SELECT product_id, product_name, category_name FROM products INNER JOIN  
categories ON products.category_id = categories.category_id;
```

Если мы выберем тот же вариант из приведенной выше таблицы продуктов, то получим следующий результат:



product_id	product_name	category_name
33	Geitost	Dairy Products
34	Sasquatch Ale	Beverages
35	Steeleye Stout	Beverages
36	Inlagd Sill	Seafood



Практика - Различия типы соединении

INNER JOIN	Возвращает записи, которые имеют совпадающие значения в обеих таблицах
LEFT JOIN	Возвращает все записи из левой таблицы и совпадающие записи из правой таблицы
RIGHT JOIN	Возвращает все записи из правой таблицы и совпадающие записи из левой таблицы
FULL JOIN	Возвращает все записи, если есть совпадение в левой или правой таблице

Практика - Внутреннее соединение

- Ключевое слово **INNER JOIN** позволяет выбрать записи, которые имеют совпадающие значения в обеих таблицах.
- Давайте рассмотрим пример с использованием нашей фиктивной таблицы testproducts:

testproduct_id	product_name	category_id
1	Johns Fruit Cake	3
2	Marys Healthy Mix	9
3	Peters Scary Stuff	10
4	Jims Secret Recipe	11
5	Elisabeths Best Apples	12
6	Janes Favorite Cheese	4
7	Billys Home Made Pizza	13
8	Ellas Special Salmon	8
9	Roberts Rich Spaghetti	5
10	Mias Popular Ice	14

(10 rows)

Практика - Внутреннее соединение

- Мы попытаемся объединить таблицу testproducts с таблицей categories:

category_id	category_name	description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

(8 rows)

Практика - Внутреннее соединение

- Обратите внимание, что многие продукты в testproducts имеют category_id, который не соответствует ни одной из категорий в таблице категорий.
- Используя INNER JOIN, мы не получим записи, в которых нет совпадений, мы получим только те записи, которые соответствуют обеим таблицам:

```
SELECT testproduct_id, product_name, category_name FROM testproducts INNER JOIN  
categories ON testproducts.category_id = categories.category_id;
```



testproduct_id	product_name	category_name
1	Johns Fruit Cake	Confections
6	Janes Favorite Cheese	Dairy Products
8	Ellas Special Salmon	Seafood
9	Roberts Rich Spaghetti	Grains/Cereals

(4 rows)

Примечание: JOIN и INNER JOIN дадут одинаковый результат.

INNER - это тип соединения по умолчанию для JOIN, поэтому, когда вы пишете JOIN, синтаксический анализатор фактически записывает INNER JOIN.

Практика - Левое соединение

- Ключевое слово **LEFT JOIN** позволяет выбрать все записи из таблицы "left", а соответствующие записи - из таблицы "right". В результате будет получено 0 записей с правой стороны, если совпадений нет.
- Давайте рассмотрим пример с использованием нашей фиктивной таблицы testproducts:

testproduct_id	product_name	category_id
1	Johns Fruit Cake	3
2	Marys Healthy Mix	9
3	Peters Scary Stuff	10
4	Jims Secret Recipe	11
5	Elisabeths Best Apples	12
6	Janes Favorite Cheese	4
7	Billys Home Made Pizza	13
8	Ellas Special Salmon	8
9	Roberts Rich Spaghetti	5
10	Mias Popular Ice	14

(10 rows)

Практика - Левое соединение

- Мы попытаемся объединить таблицу `testproducts` с таблицей `categories`:

category_id	category_name	description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

(8 rows)

Примечание: Многие продукты в `testproducts` имеют `category_id`, который не соответствует ни одной из категорий в таблице категорий.

Практика - Левое соединение

- Используя **LEFT JOIN**, мы получим все записи из testproducts, даже те, которые не соответствуют таблице категорий:

```
SELECT testproduct_id, product_name, category_name
FROM testproducts
     LEFT JOIN categories ON testproducts.category_id =
categories.category_id;
```



testproduct_id	product_name	category_name
1	Johns Fruit Cake	Confections
2	Marys Healthy Mix	
3	Peters Scary Stuff	
4	Jims Secret Recipe	
5	Elisabeths Best Apples	
6	Janes Favorite Cheese	Dairy Products
7	Billys Home Made Pizza	
8	Ellas Special Salmon	Seafood
9	Roberts Rich Spaghetti	Grains/Cereals
10	Mias Popular Ice	

(10 rows)

Примечание: LEFT JOIN и LEFT OUTER JOIN дадут одинаковый результат.

OUTER - это тип соединения по умолчанию для LEFT JOIN, поэтому, когда вы пишете LEFT JOIN, синтаксический анализатор фактически записывает LEFT OUTER JOIN.

Практика - Правое соединение

- Ключевое слово RIGHT JOIN позволяет выбрать ВСЕ записи из таблицы "right", а соответствующие записи - из таблицы "left". В случае отсутствия совпадений в результате будет получено 0 записей с левой стороны.
- Давайте рассмотрим пример с использованием нашей фиктивной таблицы testproducts:

testproduct_id	product_name	category_id
1	Johns Fruit Cake	3
2	Marys Healthy Mix	9
3	Peters Scary Stuff	10
4	Jims Secret Recipe	11
5	Elisabeths Best Apples	12
6	Janes Favorite Cheese	4
7	Billys Home Made Pizza	13
8	Ellas Special Salmon	8
9	Roberts Rich Spaghetti	5
10	Mias Popular Ice	14

(10 rows)

Практика - Правое соединение

- Мы попытаемся объединить таблицу testproducts с таблицей categories:

category_id	category_name	description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish


(8 rows)

Примечание: Многие продукты в testproducts имеют category_id, который не соответствует ни одной из категорий в таблице категорий.

Практика - Правое соединение

- Используя **RIGHT JOIN**, мы получим все записи из категорий, даже те, которые не соответствуют таблице testproducts:

```
SELECT testproduct_id, product_name, category_name
FROM testproducts
     RIGHT JOIN categories ON testproducts.category_id =
categories.category_id;
```



testproduct_id	product_name	category_name
1	Johns Fruit Cake	Confections
6	Janes Favorite Cheese	Dairy Products
8	Ellas Special Salmon	Seafood
9	Roberts Rich Spaghetti	Grains/Cereals
		Condiments
		Meat/Poultry
		Beverages
		Produce

(8 rows)

Примечание: **RIGHT JOIN** и **RIGHT OUTER JOIN** дадут одинаковый результат.

OUTER - это тип соединения по умолчанию для **RIGHT JOIN**, поэтому, когда вы пишете **RIGHT JOIN**, синтаксический анализатор фактически записывает **RIGHT OUTER JOIN**.

Практика - Полное соединение

- Ключевое слово **FULL JOIN** позволяет выбрать ВСЕ записи из обеих таблиц, даже если они не совпадают. Для строк с совпадением доступны значения из обеих таблиц, если совпадения нет, пустым полям будет присвоено значение NULL.
- Давайте рассмотрим пример с использованием нашей фиктивной таблицы testproducts:

testproduct_id	product_name	category_id
1	Johns Fruit Cake	3
2	Marys Healthy Mix	9
3	Peters Scary Stuff	10
4	Jims Secret Recipe	11
5	Elisabeths Best Apples	12
6	Janes Favorite Cheese	4
7	Billys Home Made Pizza	13
8	Ellas Special Salmon	8
9	Roberts Rich Spaghetti	5
10	Mias Popular Ice	14

(10 rows)

Практика - Полное соединение

- Мы попытаемся объединить таблицу testproducts с таблицей categories:

category_id	category_name	description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

(8 rows)

Примечание: Многие продукты в testproducts имеют category_id, который не соответствует ни одной из категорий в таблице категорий.

Практика - Полное соединение

- Используя **FULL** объединение, мы получим все записи как из таблицы `categories`, так и из таблицы `testproducts`:

```
SELECT testproduct_id, product_name, category_name
FROM testproducts
FULL JOIN categories ON testproducts.category_id =
categories.category_id;
```

testproduct_id	product_name	category_name
1	Johns Fruit Cake	Confections
2	Marys Healthy Mix	
3	Peters Scary Stuff	
4	Jims Secret Recipe	
5	Elisabeths Best Apples	
6	Janes Favorite Cheese	Dairy Products
7	Billys Home Made Pizza	
8	Ellas Special Salmon	Seafood
9	Roberts Rich Spaghetti	Grains/Cereals
10	Mias Popular Ice	
		Condiments
		Meat/Poultry
		Beverages
		Produce

(14 rows)

Примечание: FULL JOIN и FULL OUTER JOIN дадут одинаковый результат.

OUTER - это тип соединения по умолчанию для FULL JOIN, поэтому, когда вы пишете FULL JOIN, синтаксический анализатор фактически записывает ПОЛНОЕ ВНЕШНЕЕ СОЕДИНЕНИЕ.

Практика - Соединение CROSS

- Ключевое слово CROSS JOIN сопоставляет ВСЕ записи из "левой" таблицы с каждой записью из "правой" таблицы.
- Это означает, что все записи из "правой" таблицы будут возвращены для каждой записи в "левой" таблице.
- Этот способ объединения потенциально может привести к получению очень большой таблицы, и вам не следует использовать его, если в этом нет необходимости.
- Давайте рассмотрим пример с использованием нашей фиктивной таблицы testproducts:

testproduct_id	product_name	category_id
1	Johns Fruit Cake	3
2	Marys Healthy Mix	9
3	Peters Scary Stuff	10
4	Jims Secret Recipe	11
5	Elisabeths Best Apples	12
6	Janes Favorite Cheese	4
7	Billys Home Made Pizza	13
8	Ellas Special Salmon	8
9	Roberts Rich Spaghetti	5
10	Mias Popular Ice	14

(10 rows)

Практика - Соединение CROSS

- Мы попытаемся объединить таблицу testproducts с таблицей categories:

category_id	category_name	description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

(8 rows)

Практика - Соединение CROSS

- Мы попытаемся объединить таблицу testproducts с таблицей categories:

category_id	category_name	description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

(8 rows)

Практика - Соединение CROSS

Примечание: Метод **CROSS** соединения вернет ВСЕ категории для КАЖДОГО тестового продукта, что означает, что он вернет 80 строк ($10 * 8$).

```
SELECT testproduct_id, product_name, category_name
FROM testproducts
     CROSS JOIN categories;
```

testproduct_id	product_name	category_name
1	Johns Fruit Cake	Beverages
1	Johns Fruit Cake	Condiments
1	Johns Fruit Cake	Confections
1	Johns Fruit Cake	Dairy Products
1	Johns Fruit Cake	Grains/Cereals
1	Johns Fruit Cake	Meat/Poultry
1	Johns Fruit Cake	Produce
1	Johns Fruit Cake	Seafood
2	Marys Healthy Mix	Beverages
2	Marys Healthy Mix	Condiments
2	Marys Healthy Mix	Confections
2	Marys Healthy Mix	Dairy Products
2	Marys Healthy Mix	Grains/Cereals



Практика - Соединение UNION

- Оператор **UNION** используется для объединения результирующего набора из двух или более запросов.
- Запросы в объединении должны соответствовать следующим правилам:
 - Они должны содержать одинаковое количество столбцов
 - Столбцы должны содержать одинаковые типы данных
 - Столбцы должны располагаться в одинаковом порядке

```
SELECT product_id, product_name FROM products UNION SELECT testproduct_id,  
product_name FROM testproducts ORDER BY product_id;
```



Практика - Соединение UNION и UNION ALL

- С помощью оператора **UNION**, если некоторые строки в двух запросах возвращают одинаковый результат, в списке будет отображаться только одна строка, поскольку UNION выбирает только разные значения.
- Используйте **UNION ALL** для возврата повторяющихся значений.

UNION

```
SELECT product_id FROM products UNION SELECT testproduct_id FROM testproducts  
ORDER BY product_id;
```

UNION ALL

```
SELECT product_id FROM products UNION ALL SELECT testproduct_id FROM testproducts  
ORDER BY product_id;
```



Практика - Группировка

- Предложение **GROUP BY** группирует строки с одинаковыми значениями в сводные строки, например "найти количество клиентов в каждой стране".
- Предложение **GROUP BY** часто используется с агрегатными функциями, такими как **COUNT()**, **MAX()**, **MIN()**, **SUM()**, **AVG()**, для группировки результирующего набора по одному или нескольким столбцам.

```
SELECT COUNT(customer_id), country FROM customers GROUP BY country;
```

- GROUP BY с использованием JOIN

```
SELECT customers.customer_name, COUNT(orders.order_id) FROM orders LEFT JOIN  
customers ON orders.customer_id = customers.customer_id GROUP BY customer_name;
```




Практика - Оператор Having

- Предложение HAVING было добавлено в SQL, потому что предложение WHERE нельзя использовать с агрегатными функциями.
- Агрегатные функции часто используются с предложениями GROUP BY, и, добавив HAVING, мы можем написать условие, как мы это делаем с предложениями WHERE.

```
SELECT COUNT(customer_id), country FROM customers GROUP BY country
        HAVING COUNT(customer_id) > 5;
```

```
SELECT order_details.order_id, SUM(products.price) FROM order_details LEFT JOIN
        products ON
        order_details.product_id = products.product_id GROUP BY order_id
        HAVING SUM(products.price) > 400.00;
```

```
SELECT customers.customer_name, SUM(products.price) FROM order_details
        LEFT JOIN products ON order_details.product_id = products.product_id
        LEFT JOIN orders ON order_details.order_id = orders.order_id
        LEFT JOIN customers ON orders.customer_id = customers.customer_id
GROUP BY customer_name HAVING SUM(products.price) > 1000.00;
```



Практика - Оператор EXISTS

- Оператор EXISTS используется для проверки существования какой-либо записи во вложенном запросе.
- Оператор EXISTS возвращает значение TRUE, если вложенный запрос возвращает одну или несколько записей.

```
SELECT customers.customer_name FROM customers WHERE EXISTS (  
    SELECT order_id FROM orders WHERE customer_id = customers.customer_id);
```

- Результат в приведенном выше примере показал, что у 89 клиентов был по крайней мере один заказ в таблице заказов.



Практика - Оператор NOT EXIST

- Чтобы проверить, у каких клиентов нет заказов, мы можем использовать оператора NOT вместе с оператором EXISTS :

```
SELECT customers.customer_name FROM customers WHERE NOT EXISTS (  
    SELECT order_id FROM orders WHERE customer_id = customers.customer_id);
```



Практика - Оператор ANY

- Оператор ANY позволяет выполнить сравнение между значением одного столбца и диапазоном других значений.
- Оператор ANY:
 - возвращает логическое значение, в результате
 - чего возвращается значение TRUE, если какое-либо из значений вложенного запроса соответствует условию
- ANY означает, что условие будет выполнено, если операция выполнена для любого из значений в диапазоне.

```
SELECT product_name FROM products WHERE product_id = ANY (  
    SELECT product_id  
    FROM order_details  
    WHERE quantity > 120  
);
```



Практика - Оператор ALL

- Оператор ALL:
 - возвращает логическое значение, в результате
 - возвращает значение TRUE, если все значения вложенного запроса соответствуют условию
 - используется с операторами SELECT, WHERE и HAVING
- ALL означает, что условие будет выполнено, только если операция выполняется для всех значений в диапазоне.

```
SELECT product_name FROM products WHERE product_id = ALL (  
  SELECT product_id  
  FROM order_details  
  WHERE quantity > 10  
);
```



Практика - Оператор CASE

- Выражение **CASE** проходит через условия и возвращает значение, когда выполняется первое условие (например, оператор if-then-else).
- Как только условие выполняется, оно прекращает чтение и возвращает результат. Если ни одно из условий не выполняется, оно возвращает значение в предложении **ELSE**.
- Если нет элемента **ELSE** и никакие условия не выполнены, он возвращает значение **NULL**.

```
SELECT product_name ,  
       CASE  
         WHEN price < 10 THEN 'Low price product'  
         WHEN price > 50 THEN 'High price product'  
         ELSE  
           'Normal product'  
       END  
FROM products;
```



Практика - Оператор CASE

- Если для поля "case" не указано имя столбца, синтаксический анализатор использует регистр в качестве имени столбца.
- Чтобы указать имя столбца, добавьте псевдоним после ключевого слова **END**.

```
SELECT product_name ,  
       CASE  
         WHEN price < 10 THEN 'Low price product'  
         WHEN price > 50 THEN 'High price product'  
         ELSE  
           'Normal product'  
         END AS "price category"  
FROM products;
```



Ссылки

- <https://docs.oracle.com/javaee/7/api/javax/persistence/package-summary.html>
- <https://medium.com/@sendvjs/spring-data-jpa-af0f0c0c78a2>
- <https://practicum.yandex.ru/blog/cto-takoe-subd-postgresql/>
- [PostgreSQL MVCC: Общие сведения об управлении многоверсионным параллелизмом в PostgreSQL | Сатиш Мишра | Терпимая \(medium.com\)](#)
- [W3School](#)

PostgreSQL 11 Administration

Cookbook

Over 175 recipes for database administrators to manage
enterprise databases



Simon Riggs, Gianni Ciolli
and Sudheer Kumar Meesala

Packt>

www.packt.com



Thank you!
Presented by
Moxirbek Maxkamov
(mokhirbek.makhkam@gmail.com)