

Тема 1. Устройство PostgreSQL

Основные понятия

Модель данных описывает данные, определяет способ их хранения и взаимодействия с ними.

Реляционная модель данных описывает данные в форме так называемых отношений.

База данных (БД, Database) — это совокупность данных, которые хранятся определённым образом, в соответствии с той или иной моделью данных.

Схема (Schema) — это контейнер, который содержит набор таблиц, объединённых по какому-либо признаку.

Таблица (Table) — основной объект хранения данных. В таблице есть столбцы с определёнными типами данных, строки с конкретными значениями и ячейки — пересечения столбцов и строк. Столбцы в таблице БД также называют «поля» или «колонки», а строки ещё называют «записи».

Система управления базами данных (СУБД) — это комплекс программ, который позволяет создать базу данных, наполнить её новыми таблицами, отобразить содержимое, редактировать существующие таблицы.

PostgreSQL относится к **клиент-серверным** СУБД. В клиент-серверной архитектуре СУБД есть два основных компонента:

1. **Клиент** — приложение, которому необходимо выполнить какие-то операции в БД.
2. **Сервер** — устройство, которое управляет файлами БД, принимает подключения клиентских приложений и выполняет запросы клиентов к базам данных.

Клиентами могут быть, например:

- **Терминальные приложения** — например, `psql`: командная строка для СУБД, где всё взаимодействие происходит в текстовом виде.
- **Менеджеры БД** с графическим интерфейсом (графические клиенты) — например, `pgAdmin`, `DBeaver`.

Тема 1. Устройство PostgreSQL

Основные возможности популярных клиентов:

	psql	pgAdmin	DBeaver
С какой РСУБД работает	PostgreSQL	PostgreSQL	множество
Возможность запуска в терминале	есть	нет	нет
Графическая оболочка	нет	есть	есть
Возможность подключаться к нескольким СУБД одновременно	нет	есть	есть

Тема 2. Основы SQL.DDL

Основные понятия

SQL — язык программирования, предназначенный для управления данными в реляционной базе данных.

DDL (Data Definition Language) — язык описания данных. Команды DDL позволяют создавать, изменять и удалять объекты баз данных. Основные операторы DDL: **CREATE, ALTER, DROP**.

DML (Data Manipulation Language) — язык манипулирования данными. Команды DML позволяют изменять данные, хранящиеся в таблицах. Основные операторы DML: **INSERT, UPDATE, DELETE**.

DQL (Data Query Language) — язык запроса данных. Команды DQL позволяют извлекать данные из базы. Основные операторы DQL: **SELECT, JOIN, UNION, GROUP BY, HAVING** и другие.

TCL (Transaction Control Language) — язык управления транзакциями. Основные операторы TCL: **COMMIT, ROLLBACK, SAVEPOINT**.

DCL (Data Control Language) — язык управления доступа к данным. Основные операторы DCL: **GRANT, REVOKE**.

Основные операторы DDL

CREATE — создание БД, схемы или таблицы:

```
CREATE TABLE схема.имя_таблицы (
    имя_поля_1 тип_поля_1,
    имя_поля_2 тип_поля_2,
    ...
    имя_поля_n тип_поля_n
);
```

ALTER — изменение объектов:

```
ALTER TABLE имя_таблицы
    ADD/DROP/ALTER -- действие
    COLUMN/CONSTRAINT -- компонент таблицы
    параметры -- в случае добавления или изменения компонента.
                -- DROP выполняется без параметров;
```

Тема 2. Основы SQL.DDL

DROP — удаление любых объектов: таблиц, схем и целых БД.

```
DROP ТИП_ОБЪЕКТА имя_объекта;
```

Базовые типы данных

Числа

Тип данных	Псевдоним	Что хранит	Пример
<code>integer</code>	<code>int</code> , <code>int4</code>	Целые числа	Количество смартфонов на складе Население Индии
<code>bigint</code>	<code>int8</code>	Большие целые числа	Счётчик строк в гигантских таблицах
<code>smallint</code>	<code>int2</code>	Маленькие целые числа	Вес колибри
<code>real</code>	-	Числа с переменной точностью	Межзвёздные расстояния
<code>double precision</code>	-	Большие числа с переменной точностью	Количество атомов во Вселенной
<code>numeric</code>	<code>decimal</code>	Числа с заданной точностью и большим кол-вом цифр	Выручка за месяц Расчёт отпускных

Тема 2. Основы SQL.DDL

Базовые типы данных

Символы

<code>text</code>		Строки без ограничения длины	Короткие заметки «Война и мир»
<code>character(n)</code>	<code>char(n)</code>	Строки фиксированной длины n	Автомобильные номера
<code>character varying(n)</code>	<code>varchar(n)</code>	Строки переменной длины с максимальной длиной n	Сообщения в Twitter

Дата и время

Тип данных	Псевдоним	Что хранит	Пример
<code>date</code>		<p>Дата без времени Стандарт: <code>yyyy-mm-dd</code> Можно и так: <code>dd.mm.yyyy</code> <code>dd/mm/yyyy</code></p>	<p>Дата рождения 23.10.1985</p>
<code>time</code>		<p>Время без даты Стандарт: <code>hh:mm:ss</code></p>	<p>Часы работы магазина открытие: <code>10:00</code> закрытие: <code>19:30</code></p> <p>Регистрации времени забега участников марафона <code>02:01:09</code></p> <p>Время взмаха крыльев комара <code>00:00:00,001</code></p>

Тема 2. Основы SQL.DDL

Базовые типы данных

Дата и время (продолжение)

Тип данных	Псевдоним	Что хранит	Пример
<code>timestamp</code>		Дата и время Стандарт: <code>yyyy-mm-dd hh:mm:ss</code>	Регистрация входящего телефонного звонка
<code>timestamp</code> <code>with time zone</code>	<code>timestamptz</code>	Дата и время с часовым поясом <code>yyyy-mm-dd hh:mm:ss+00</code>	Дата и время «по Москве» <code>2004-12-31 10:23:54+03</code>

Логические значения

Тип данных	Псевдоним	Что хранит	Пример
<code>boolean</code>	<code>bool</code>	Значения «истина» и «ложь»	<code>true</code> и аналоги: <code>'true'</code> , <code>'t'</code> , <code>'yes'</code> , <code>'y'</code> , <code>'on'</code> , <code>1</code> <code>false</code> и аналоги: <code>'false'</code> , <code>'f'</code> , <code>'no'</code> , <code>'n'</code> , <code>'off'</code> , <code>0</code>

CONSTRAINT — ограничения

NOT NULL — запрещает отсутствующие значения.

DEFAULT — задаёт значение по умолчанию.

CHECK — проверяет условие.

UNIQUE — запрещает повторение значений в колонке.

Тема 3. Основы SQL.DML

UPDATE — обновление существующих записей таблицы:

```
UPDATE имя_таблицы
SET
-- список обновляемых полей и их новых значений через запятую
-- в таком формате
поле_1 = значение_1,
поле_2 = значение_2,
-- ...
поле_n = значение_n
WHERE условия;
```

DELETE — удаление записей:

```
DELETE FROM имя_таблицы
WHERE условие;
```

INSERT — внесение данных в таблицы, то есть создание новых записей (строк):

```
INSERT INTO имя_таблицы (столбец_1, столбец_2, ... столбец_n)
VALUES (значение_1, значение_2, ... значение_n);
```

INSERT INTO ... SELECT — сохранение результата выборки для дальнейшего использования:

```
INSERT INTO имя_таблицы_1 (
поле_1.таблица_1,
поле_2.таблица_1,
-- --
поле_n.таблица_1
) SELECT
поле_1.таблица_2,
поле_2.таблица_2,
-- --
поле_n.таблица_2
FROM имя_таблицы_2
-- только в случае, если вы переносите не все строки из таблицы 2 в таблицу 1
WHERE условия;
```

Тема 3. Основы SQL.DML

ON CONFLICT (<ограничение>) DO ... — предотвращение конфликтов при вставке данных, которые противоречат наложенным ограничениям:

- **ON CONFLICT (<ограничение>) DO NOTHING** — в случае конфликта ничего не произойдёт;
- **ON CONFLICT (<ограничение>) DO UPDATE SET ...** — в случае конфликта либо выполнится команда **INSERT** и новое значение добавится в таблицу, либо значение в таблице обновится.

SELECT — чтение данных:

```
-- выбор определённых полей из таблицы
SELECT поле_1,
       поле_2,
       поле_3 ...
FROM таблица;
```

CAST — изменение типа данных поля при выгрузке:

```
-- изменение типа данных поля при выгрузке
SELECT CAST(поле AS тип_данных)
FROM таблица;
```

WHERE — фильтрация данных по условию:

```
-- фильтрация данных по условию
SELECT поле_1,
       поле_2 -- выбор полей
FROM таблица -- таблица, из которой выгружают данные
WHERE условие; -- условие для среза данных
```

BETWEEN ... AND — фильтрация, в которой значение в поле_1 находится между значением_1 и значением_2 включительно:

```
/*
фильтрация, в которой значение в поле_1 находится
между значением_1 и значением_2 включительно
*/
SELECT *
FROM таблица
WHERE поле_1 BETWEEN значение_1 AND значение_2;
```

Тема 3. Основы SQL.DML

IN — фильтрация, в которой все значения поля находятся в списке:

```
-- фильтрация, в которой все значения поля находятся в списке
SELECT *
FROM таблица
WHERE поле IN ('значение_1', 'значение_2', 'значение_3');
```

NULL

```
-- выбор записей с пропусками в поле
SELECT *
FROM таблица
WHERE поле IS NULL;
```

```
-- выбор записей без пропусков в поле
SELECT *
FROM таблица
WHERE поле IS NOT NULL;
```

CASE — действия в зависимости от условий:

```
-- действия в зависимости от условий
CASE
    WHEN условие_1 THEN результат_1
    WHEN условие_2 THEN результат_2
    WHEN условие_3 THEN результат_3
    ELSE результат_4
END;
```

Тема 3. Основы SQL.DML

LIKE, NOT LIKE, ILIKE — нахождение строки по шаблону.

NOT LIKE противоположен **LIKE**.

NOT LIKE и **LIKE** учитывают регистр, **ILIKE** — нет.

Знак **%** показывает, какую позицию в строке занимает шаблон.

Синтаксис	Что сделает оператор
'%SQL%'	найдёт строки, которые содержат слово "SQL", где угодно внутри строки
'SQL%'	найдёт строки, которые начинаются с подстроки "SQL"
'%SQL'	найдёт строки, которые заканчиваются подстрокой "SQL"

```
строка LIKE шаблон;
строка NOT LIKE шаблон;
строка ILIKE шаблон;
```

ORDER BY — сортировка:

```
-- определяет порядок выдачи строк
SELECT столбец_1, столбец_2, ... столбец_n
FROM имя_таблицы
WHERE условие -- необязательно
ORDER BY параметр_сортировки_1, параметр_сортировки_2,
        ... параметр_сортировки_n;
```

LIMIT — ограничение выборки максимальным количеством строк:

```
SELECT name, price
FROM products
-- отсортирует по убыванию цены, значения NULL в конце
ORDER BY price DESC NULLS LAST
-- из отсортированного списка возьмёт три первых строки
LIMIT 3;

SELECT name, birthday
FROM users
ORDER BY birthday ASC -- отсортирует пользователей по дате рождения
LIMIT 1 -- возьмёт одного старейшего пользователя;
```

Тема 3. Основы SQL.DML

OFFSET K — смещение выборки:

```
SELECT name, price
FROM products
-- отфильтрует товары, в которых цена не указана
WHERE price IS NOT NULL
-- отсортирует по возрастанию цены
ORDER BY price ASC
-- из отсортированного списка выведет все товары за исключением
-- первых десяти
OFFSET 10;
```

LIMIT N OFFSET K — пагинация (разбивка на страницы):

```
SELECT name, description
FROM products
ORDER BY name ASC -- отсортирует по имени
OFFSET 30 -- пропустит первые 30 строк, показанных на первой странице
LIMIT 30 -- возьмёт следующие 30 записей для второй страницы
```

DISTINCT — удаление дубликатов:

```
SELECT DISTINCT category -- оставит в выборке только уникальные значения
FROM products;
```

DISTINCT ON — задание уникальности по перечисленным колонкам:

```
SELECT DISTINCT ON (category) category, price
FROM products
ORDER BY category ASC, price DESC;
```

Тема 3. Основы SQL.DML

Дополнительные операторы

Категория оператора	Запись в SQL	Что означает
Операторы сравнения	=	равно
	<> , !=	не равно
	>	больше
	<	меньше
	>=	больше или равно
	<=	меньше или равно
Операторы для NULL	IS NULL , ISNULL	значение отсутствует
	IS NOT NULL , NOTNULL	присутствует любое значение
Логические операторы	NOT	вернёт истинное значение (true), если все условия, которые объединяет этот оператор, истинны
	AND	вернёт истинное значение (true), если хотя бы одно из условий, которые объединяет этот оператор, истинно
	OR	меняет значение логического выражения с истинного (true) на ложное (false) и наоборот

Тема 4. Нормализация. Взаимоотношения между таблицами.

Основные понятия

Нормализация — это изменение структуры базы данных, ведущее к уменьшению избыточности данных.

Нормальная форма — набор требований, который предъявляется к отношению.

Ключ — это столбец или несколько столбцов, таких, что не существует двух строк с одинаковыми значениями этих столбцов.

Неключевые столбцы — столбцы, не входящие ни в один из потенциальных ключей.

Внешний ключ — поле, которое отсылает к первичному ключу другой таблицы.

1НФ — таблица находится в 1НФ, если строки в ней не дублируются, и в каждой её ячейке содержится только одно значение.

2НФ — таблица находится в 1НФ, и каждый неключевой атрибут зависит от всего ключа, а не от его части.

3НФ — таблица находится в 2НФ, неключевые поля зависят только от ключа и не зависят от других неключевых полей.

Типы связей:

- «**один к одному**» — одна запись в первой таблице связана с одной записью во второй таблице.
- «**один ко многим**» — одна запись в одной таблице соответствует нескольким записям в другой таблице.
- «**многие ко многим**» — одна запись одной таблицы соответствует нескольким записям другой таблицы и наоборот.

ER-диаграмма — диаграмма, иллюстрирующая устройство базы данных с учётом таблиц и связей между ними.

Тема 5. Связанные таблицы

Декартово произведение таблиц:

```
-- декартово произведение таблиц  
SELECT * FROM clients, products;
```

Псевдонимы:

```
-- назначение псевдонима при выгрузке  
SELECT поле AS новое_название_поля  
-- упрощённый способ назначения псевдонима в PostgreSQL  
SELECT поле новое_название_поля
```

INNER JOIN — внутреннее соединение таблиц:

```
-- внутреннее соединение таблиц  
SELECT таблица_1.поле_1 AS поле_1,  
       таблица_2.поле_2 AS поле_2  
FROM таблица_1  
INNER JOIN таблица_2 ON таблица_1.поле_1 = таблица_2.поле_2;  
-- INNER JOIN можно заменить на JOIN
```

LEFT JOIN — левое внешнее соединение таблиц:

```
-- левое внешнее соединение таблиц  
SELECT таблица_1.поле_1 AS поле_1,  
       таблица_2.поле_2 AS поле_2  
FROM таблица_1  
LEFT JOIN таблица_2 ON таблица_1.поле_1 = таблица_2.поле_2;
```

RIGHT JOIN — правое внешнее соединение таблиц:

```
-- правое внешнее соединение таблиц  
SELECT таблица_1.поле_1 AS поле_1,  
       таблица_2.поле_2 AS поле_2  
FROM таблица_1  
RIGHT JOIN таблица_2 ON таблица_1.поле_1 = таблица_2.поле_2;  
-- ВАЖНО! Правое присоединение можно поменять на левое,  
-- просто поменяв таблицы местами
```

Тема 5. Связанные таблицы

Присоединение нескольких таблиц:

```
-- присоединение нескольких таблиц
SELECT таблица_1.поле_1 AS поле_1,
       таблица_2.поле_2 AS поле_2
       таблица_3.поле_3 AS поле_3
FROM таблица_1
JOIN таблица_2 ON таблица_1.поле_1 = таблица_2.поле_2
JOIN таблица_3 ON таблица_1.поле_1 = таблица_3.поле_3;
```

UNION и **UNION ALL** — объединение запросов:

```
-- объединение запросов
SELECT поле_1
FROM таблица_1
UNION --( или UNION ALL) -- UNION ALL оставляет дубликаты
SELECT поле_2
FROM таблица_2;
```

USING:

```
-- сокращённая форма соединения, если названия столбцов для соединения совпадают
SELECT *
FROM left_table
[уточнение типа соединения] JOIN right_table
USING (столбцы для соединения)
```

NATURAL:

```
-- сокращённая форма соединения, если названия столбцов для соединения совпадают
SELECT * FROM clients NATURAL JOIN purchases;
```

INTERSECT — нахождение общих строк в результатах двух или более выборок:

```
-- нахождение общих строк в результатах двух или более выборок
SELECT product_name FROM products_in_stock
INTERSECT
SELECT product_name FROM popular_products;
```

EXCEPT — вычитание строк одного запроса из другого:

```
-- вычитание строк одного запроса из другого
SELECT product_name FROM popular_products
EXCEPT
SELECT product_name FROM products_in_stock;
```

Тема 6. Группировка данных, агрегирующие функции

Агрегирующие функции

Функция	Что возвращает
COUNT(*)	число записей в таблице
COUNT(column)	число записей в поле <code>column</code>
COUNT(DISTINCT column)	количество уникальных значений в поле <code>column</code>
SUM(column)	сумма значений в поле
AVG(column)	среднее значений в поле
MIN(column)	минимум значений в поле
MAX(column)	максимум значений в поле

Синтаксис:

```
SELECT  
COUNT(*)  
FROM таблица;
```

Тема 6. Группировка данных, агрегирующие функции

Математические функции

Функция	Описание	Пример	Результат
ABS	Возвращает модуль числа	ABS(-14)	14
CEILING	Возвращает число, округлённое до целого в большую сторону	CEILING(42.8)	43
FLOOR	Возвращает число, округлённое до целого в меньшую сторону	FLOOR(42.8)	42
ROUND	Округляет значение до ближайшего числа, а при указании второго аргумента округляет число до определённого количества знаков после запятой	ROUND(42.4) ROUND(42.4382, 2)	42 42.44
TRUNC	Усекает значение до ближайшего числа или до указанного количества знаков после запятой, но число при этом не округляет	TRUNC(42.6) TRUNC(42.6382, 2)	42 42.63
POWER	Возвращает число, возведённое в степень, — нужную степень указывают вторым аргументом	POWER(9, 3)	729
SQRT	Извлекает квадратный корень из числа	SQRT(9)	3

Тема 6. Группировка данных, агрегирующие функции

Функции работы со строками

Название	Что делает
Целые строки	
LENGTH	Подсчитывает количество символов в строке.
LOWER	Изменяет регистр символа в строке на нижний.
UPPER	Изменяет регистр символа в строке на верхний.
INITCAP	Переводит первую букву каждого слова в строке в верхний регистр, а остальные — в нижний.
CONCAT	Объединяет несколько строк в одну с несколькими разделителями.
CONCAT_WS	Объединяет несколько строк в одну с одним разделителем.
	Объединяет несколько строк в одну.
STRING_AGG	Соединяет нескольких строк в одну при агрегации данных.
Подстроки	
TRIM	Удаляет с обеих сторон строки наибольшую подстроку, которая содержит определённые символы.
LTRIM	Удаляет символы слева в строке.
RTRIM	Удаляет символы справа в строке.
REPLACE	Заменяет один символ на другой.
SPLIT_PART	Разделяет строку по заданному символу-разделителю и выдаёт определённую <i>n</i> -подстроку, которую необходимо извлечь.
SUBSTR	Извлекает необходимую информацию определённой длины с заданной позиции.
STRPOS POSITION	Показывает начальную позицию первого вхождения подстроки в строке.

Тема 6. Группировка данных, агрегирующие функции

Группировка

GROUP BY — группировка данных:

```
-- группировка данных
SELECT поле_1,
       поле_2,
       поле_3,
       АГРЕГИРУЮЩАЯ_ФУНКЦИЯ(поле)
FROM таблица
GROUP BY поле_1,
       поле_2,
       поле_3;
-- ВАЖНО! Сколько полей без агрегации в SELECT, столько и должно быть в GROUP BY
```

HAVING — срез после группировки:

```
-- срез после группировки
SELECT поле_1,
       поле_2,
       поле_3,
       АГРЕГИРУЮЩАЯ_ФУНКЦИЯ(поле)
FROM таблица
GROUP BY поле_1
HAVING АГРЕГИРУЮЩАЯ_ФУНКЦИЯ(поле) > n;
-- ВАЖНО! WHERE работает только с изначальными данными,
-- HAVING — только с агрегированными
```

ORDER BY с агрегирующими функциями:

```
-- сортировка данных
SELECT поле_1,
       поле_2,
       поле_3,
       АГРЕГИРУЮЩАЯ_ФУНКЦИЯ(поле)
FROM таблица
ORDER BY поле_1, -- сортировка по возрастанию
         поле_2 DESC, -- сортировка по убыванию
         поле_3 ASC; -- сортировка по возрастанию
```

Тема 6. Группировка данных, агрегирующие функции

DATE_TRUNC — усечение даты до части:

```
-- усечение даты до части  
SELECT DATE_TRUNC('часть_даты_до_которой_усекаем', поле) AS новое_поле_с_датой  
FROM таблица;
```

Параметры функции **DATE_TRUNC**, одинарные кавычки обязательны :

- 'millennium' — тысячелетие.
- 'century' — век.
- 'decade' — десятилетие.
- 'year' — год.
- 'quarter' — квартал.
- 'month' — месяц.
- 'week' — неделя.
- 'day' — день.
- 'hour' — час.
- 'minute' — минута.
- 'second' — секунда.
- 'microseconds' — микросекунды.
- 'milliseconds' — миллисекунды.

Тема 6. Группировка данных, агрегирующие функции

EXTRACT — извлечение части даты:

```
-- извлечение части даты
SELECT EXTRACT(часть_даты FROM поле) AS новое_поле_с_датой
FROM таблица;
```

Параметры функции **EXTRACT**, кавычки не нужны :

- **century** — век.
- **year** — год.
- **quarter** — квартал.
- **month** — месяц.
- **week** — неделя.
- **day** — день года.
- **doy** — день года, число от 1 до 365 или 366, если год високосный.
- **dow** — день недели, число от 0 до 6, где понедельник: 1, воскресенье: 0.
- **isodow** — день недели, число от 1 до 7, где понедельник: 1, воскресенье: 7.
- **hour** — час.
- **minute** — минута.
- **second** — секунда.
- **milliseconds** — миллисекунды.

Используя функции **EXTRACT** и **DATE_TRUNC**, приводите поле к типу **timestamp**, чтобы решить проблему с часовыми поясами в PostgreSQL:

```
SELECT EXTRACT(MONTH FROM CAST(поле AS timestamp)) AS первое_поле_с_датой,
       DATE_TRUNC('month', CAST(поле AS timestamp)) AS второе_поле_с_датой
  FROM таблица;
```

Тема 6. Группировка данных, агрегирующие функции

Генерация последовательностей значений

`GENERATE_SERIES(начало_диапазона, конец_диапазона [, шаг_значений])` — возвращает множество строк с определённой последовательностью значений в заданном диапазоне.

Генерация множества строк с определённой последовательностью чисел:

```
SELECT
    GENERATE_SERIES(0.1, 5, 1) AS seq_1,
    GENERATE_SERIES(15, 0, -2.41) AS seq_2;
```

Генерация множества строковых значений:

```
SELECT
    CHR(ASCII('A') + GENERATE_SERIES(0, 4)) AS eng,
    CHR(ASCII('Я') - GENERATE_SERIES(0, 8, 2)) AS rus;
```

Генерация множества с последовательностью дат или временных интервалов:

```
SELECT *
FROM GENERATE_SERIES('2023-05-01 00:00'::timestamp, '2023-05-03 12:00', '12 hours')
```

Генерация нескольких множеств, у которых есть перекрёстные взаимоотношения:

```
SELECT
    CHR(ASCII('A') + a) AS first_level,
    b AS second_level
FROM
    GENERATE_SERIES(0, 2, 1) AS a,
    GENERATE_SERIES(10, 13, 1) AS b;
```

Создание множества строк со случайными значениями:

```
SELECT ROUND(RANDOM()*100) AS random_value
FROM GENERATE_SERIES(1, 5, 1) AS seq
```

Тема 6. Группировка данных, агрегирующие функции

Генерация последовательности случайных значений даты и времени:

```
SELECT '2023-05-01 00:00'::timestamp + RANDOM() * ('24 hours'::interval)
AS random_time
FROM GENERATE_SERIES(1, 5);
```

Генерация последовательности значений при агрегации данных:

```
SELECT movie_year, COUNT(m.film_id) AS movie_num
FROM GENERATE_SERIES(2000, 2010, 1) AS movie_year
LEFT JOIN movie AS m ON movie_year = m.release_year
-- присоединит таблицу с данными о фильмах
GROUP BY movie_year
-- сгруппирует данные по году выпуска фильма
ORDER BY movie_year
-- отсортирует вывод в порядке возрастания года выпуска фильма
```