




Что такое SQL?

 **SQL – это формальный язык описания запросов к базам данных.**

 *Structured Query Language, произносится «эс-кью-эль», реже «сиквел»; второй вариант произношения остался со времён предыдущего названия языка, «SEQUEL» – Structured English Query Language*

Практически все распространенные СУБД поддерживают SQL.

Это инструмент, с помощью которого аналитик обращается к базам данных, создает выборки, анализирует и преобразует их. В широком смысле SQL не является языком программирования и похож на простую английскую речь, которая структурирована определенным образом.



SQL – это набор операторов, которые можно использовать для описания, модификации, управления данными, а также для управления транзакциями.



Как создавать запросы в базу данных?

Оператор SELECT

AS: Алиасы (операция переименования)

MOD – взятие остатка

EXTRACT

Особенное значение NULL

DISTINCT

ORDER BY

DISTINCT и ORDER BY

Оператор SELECT

- осуществляет выборку из базы данных
- основной оператор для работы с данными
- самый сложный среди всех операторов языка SQL

Запросы для выборки из базы данных определяются командой

```
SELECT <атрибут1>, <атрибут2>, ... FROM  
<таблица>;
```

Вот как будет выглядеть создание выборки студентов с кодом группы (group_code) из таблицы со списком студентов (students):

```
postgres=# select name, group_code from students;
```

name	group_code
Иванов Иван Иванович	M80-101A-19
Васильев Василий Васильевич	M80-102A-19
Федоров Федор Федорович	M80-103A-19
Григорьев Григорий Григорьевич	M80-104A-19
Петров Петр Петрович	M80-105A-19

💬 В целом SQL нечувствителен к регистру: команды и названия атрибутов можно писать как заглавными, так и прописными буквами. Но иногда при скачивании данных из внешних систем можно встретить разные регистры или пробелы в названии столбцов и таблиц, и могут возникнуть ошибки при запросе данных из них. В таком случае регистр надо учесть с помощью использования двойных кавычек.

Символ *

Символ «*» обозначает всю схему исходного отношения. Синтаксис выглядит так:

```
SELECT * FROM <отношение>;
```

Результатом запроса будут все записи в таблице:

```
postgres=# select * from students;
```

student_id	name	group_code	birthday
1	Иванов Иван Иванович	M80-101A-19	2001-05-20
2	Васильев Василий Васильевич	M80-102A-19	2000-12-02
3	Федоров Федор Федорович	M80-103A-19	2001-01-17
4	Григорьев Григорий Григорьевич	M80-104A-19	2001-05-20
5	Петров Петр Петрович	M80-105A-19	2000-12-31

👉 Результат выполнения запроса – отношение.

Поэтому в запросе выше вместо в части <отношение> можно написать еще один запрос.

AS: Алиасы (операция переименования)

Чтобы временно переименовать таблицу или столбец, можно присвоить ей alias (псевдоним). Для этого используется ключевое слово as:

```
SELECT <атрибут1> AS <новое имя>, ...  
FROM <таблица>;
```

Поменяем наименование “код группы” (group_code) на “группу” (group):

```
postgres=# select name, group_code as group from students;  
      name      | group  
-----+-----  
Иванов Иван Иванович | M80-101A-19  
Васильев Василий Васильевич | M80-102A-19  
Федоров Федор Федорович | M80-103A-19  
Григорьев Григорий Григорьевич | M80-104A-19  
Петров Петр Петрович | M80-105A-19
```

👉 Слово as можно опускать и сразу после атрибута писать новое имя. Не стоит называть поля словами из языка SQL (например, distinct, from, as и т.д.). Если всё-таки нужно это сделать, такие названия лучше заключать в двойные кавычки.

В списке select вместо атрибутов могут использоваться выражения, например, константные значения:

```
SELECT <выражение> AS <имя>, ...  
FROM <таблица>;
```

```
postgres=# select name, 'студент' as type from students;  
      name      | type  
-----+-----  
Иванов Иван Иванович | студент  
Васильев Василий Васильевич | студент  
Федоров Федор Федорович | студент  
Григорьев Григорий Григорьевич | студент  
Петров Петр Петрович | студент
```

MOD – взятие остатка

Взятие остатка от деления a на b можно записать так:

```
SELECT MOD <a, b>, ...FROM <таблица>;
```

EXTRACT

Функция EXTRACT извлекает и возвращает отдельные части из даты (год, месяц, день) или даты-времени.

```
SELECT EXTRACT(day from <дата>) FROM  
<таблица>;
```

Слева в параметрах функции та часть, которую нужно вычленить из даты (в этом случае — день), а справа сама дата.

👉 Для округления времени существует полезная функция DATE_TRUNC.

```
SELECT date_trunc('hour', TIMESTAMP '2001-02-16 20:38:40');
```

Результат: 2001-02-16 20:00:00

```
SELECT date_trunc('year', TIMESTAMP '2001-02-16 20:38:40');
```

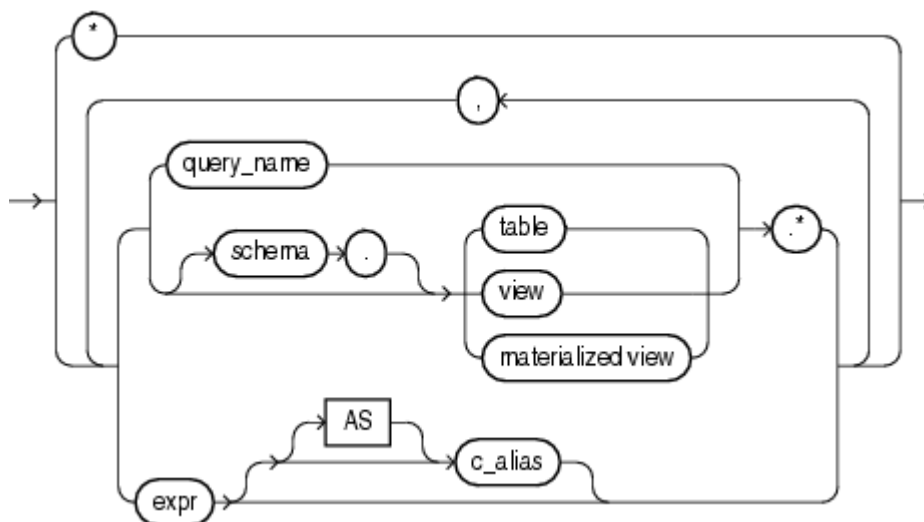
Результат: 2001-01-01 00:00:00

💻 Пример

Нужно написать запрос, который выводит id студента, имя, дату рождения и остаток от деления id на 5. Остатку от деления дать осмысленное название.

```
SELECT id, name, birthday,  
       MOD(id, 5) AS reminder  
FROM students;
```

Общий вид списка SELECT



[Бусиничное представление синтаксиса select-списка](#)

Особенное значение NULL

- NULL – отсутствие значения вообще
- Не равно никакому другому значению
- Не равно даже NULL
- Отображается как пустое поле
- \pset null 'NULL' – можно добавить в .psqlrc
- Можно использовать COALESCE(<атрибут>, <значение>)

📌 **Наличие NULL в реляционных базах данных необходимо.**

В SQL NULL — это термин, используемый для представления отсутствующего значения. NULL означает отсутствие информации, поэтому NULL не равно никакому другому значению – ни FALSE, ни пустой строке, ни нулю. NULL не равен даже NULL – ведь неизвестно, что с чем сравнивать.

💬 Значение NULL отображается как пустое. Это не всегда удобно, так как невозможно отличить, например, отсутствующее значение от пустой строки или от поля, состоящего только из пробелов.

В psql можно задать способ отображения отсутствующих значений:


\pset null 'NULL'

👉 Если пользуетесь psql: можно добавить эту команду в .psqlrc для автоматического выполнения при запуске клиента.

можно использовать функцию **COALESCE**, которая конвертирует NULL-значения в значения, заданные вторым параметром, оставляя другие значения неизменными.

Отличие варианта с pset от coalesce в том, что pset – это просто форматирование вывода, то есть только то, как вы воспринимаете информацию, в то время как coalesce влияет на данные, то есть меняет восприятие значений самой базой данных.

DISTINCT

 DISTINCT используется для удаления дубликатов из результирующего набора SELECT.

- Оставляет только уникальные значения в результирующем наборе
- Одно выражение – уникальные значения для выражения
- Несколько выражений – уникальные комбинации для выражений
- Если есть NULL – будет и в результатах


```
SELECT DISTINCT<атрибут1>, <атрибут2>, ...  
FROM <таблица>;
```

DISTINCT – это предикат, относящийся к результирующему набору данных. Вместо DISTINCT может быть указан квантор всеобщности ALL, который является значением по умолчанию и опускается.

Уникальные значения, которые будет возвращать DISTINCT, зависят от количества выражений, которые в нем содержатся.

Одно выражение — запрос будет возвращать уникальные значения для этого выражения.

Более одного выражения — уникальные комбинации для перечисленных выражений.


 Оператор DISTINCT не игнорирует NULL значения, поэтому результирующий набор будет включать в себя NULL в качестве отдельного значения.

Если в исходном наборе данных несколько NULL, они будут отнесены в одну группу.

ORDER BY


- порядок сортировки должен быть явным
- ASC|DESC
- порядковые номера вместо списка атрибутов

Если выполнить несколько раз один и тот же SELECT, порядок строк в результатах идентичен. Однако полагаться на такое совпадение нельзя, поскольку порядок может измениться. При выполнении простой выборки из таблицы СУБД не гарантирует никакого конкретного порядка вывода строк.

 Когда требуется каким-то образом **упорядочить расположение** выводимых строк, используется предложение **ORDER BY** команды SELECT.


Процесс упорядочивания начинается с самого правого атрибута списка и заканчивается самым левым. Это означает, что в результате мы получим строки, отсортированные по самому левому атрибуту, а потом по остальным.


К каждому атрибуту можно добавить признак порядка: ASC для возрастающего, DESC для убывающего.

 По умолчанию записи упорядочиваются по возрастанию значения.

Сортировку можно выполнять даже по атрибутам, отсутствующим в списке SELECT. Но естественно, что эти атрибуты должны существовать в отношении, из которого производится выборка.

```
SELECT <список_атрибутов> FROM таблица  
ORDER BY <список_атрибутов_сортировки> ASC;
```

 Вместо имени в списке можно задавать номер атрибута из списка выбора, если, например, столбец задан агрегатной функцией. Однако при этом возникают неприятности при отладке программы: любое изменение списка выбора чревато ошибкой в сортировке.\

 Параметром сортировки может быть номер столбца, при этом номер **нельзя использовать в выражении**.

Сортировка по первому столбцу запроса (name):


```
[postgres=> select name, student_id from students order by 1;
          name | student_id
-----+-----
Алехин Роман Мечиславович |         36
Башкатов Мстислав Миронович |         40
Бершов Феликс Давидович |         51
Бойцов Агап Натанович |         52
Брызгин Терентий Григорьевич |         20
Бутусов Леонид Эрнстович |         27
```

Если же 1 используется в выражении, сортировка теряет смысл (выполняется сортировка по константе, а не по столбцу):

```
[postgres=> select name from students order by 1+3;
          name
-----
Иванов Иван Иванович
Васильев Василий Васильевич
Федоров Федор Федорович
Григорьев Григорий Григорьевич
Петров Петр Петрович
Игнатьев Игнат Игнатьевич
```

В таких случаях можно использовать имена атрибутов:

```
[postgres=> select name, student_id from students order by student_id+3;
          name | student_id
-----+-----
Иванов Иван Иванович |          1
Васильев Василий Васильевич |          2
Федоров Федор Федорович |          3
Григорьев Григорий Григорьевич |          4
Петров Петр Петрович |          5
```

Особый случай возникает, когда атрибут принимает значение NULL. Стандарт упорядочивания не определяет его положение в списке, в некоторых реализациях считается, что это значение наименьшее, в других – что наибольшее.

DISTINCT и ORDER BY

Несмотря на то, что сортировку можно выполнять даже по атрибутам, отсутствующим в списке SELECT, при одновременном использовании

DISTINCT и ORDER BY возникает неоднозначность – для дублирующих строк нельзя определить, какое выбирать значение для сортировки (DISTINCT выполняется раньше ORDER BY и не оставляет информации по столбцам, отсутствующим в select-листе).

Пример ошибки:

```
SELECT DISTINCT name, birthday FROM students ORDER BY student_id;
```

ERROR: for SELECT DISTINCT, ORDER BY expressions must appear in select list.



Предикаты

NOT, AND, OR

WHERE

BETWEEN

Предикат IN

Предикат LIKE

NOT, AND, OR

- =, >, <, >=, <=, <>
- NOT, AND, OR
- IS и порядок выполнения «(» и «)»
- TRUE (истина), FALSE (ложь) или UNKNOWN (неизвестно)

Предикаты представляют собой выражения, принимающие истинностное значение. Они могут представлять собой как одно выражение, так и любую комбинацию из неограниченного количества выражений, построенную с помощью булевых операторов AND, OR или NOT.

В этих комбинациях может использоваться SQL-оператор IS, а также круглые скобки для конкретизации порядка выполнения операций.

📌 **Для предиката сравнения выражения должны быть совместимы по типам.**

Предикат в языке SQL может принимать одно из трех значений:

- TRUE (истина),
- FALSE (ложь)
- UNKNOWN (неизвестно)

Порядок выполнения

Как и арифметические операторы, при отсутствии скобок логические операторы выполняются в соответствии с их старшинством.

1. NOT
2. AND
3. OR

Одноместная операция NOT имеет наивысший приоритет.

Следующий приоритет имеет оператор AND.

```
where (
(
metric_name in (
'messenger_visitors',
'messenger_dau',
'chats_deleted',
'phone_views_in_chat',
'users_phone_views_in_chat',
'users_first_messages',
'booking_created_short_rent',
'answered_first_messages',
'delivery_order_created',
'active_items',
'active_items_with_msg',
'chats_deleted',
'first_response_messages',
'items_sold_opavito',
'items_closed_by_user_total',
'item_views',
'write_button_clicks_grouped'
)
and participant_type = 'user'
)
or
(
participant_type = 'visitor'
and metric_name in (
'first_messages',
'write_button_clicks',
'users_write_button_clicks',
'phone_views_total'
)
)
)
and is_participant_authorized = -1
and region_id = -1
and user_segment = 'Any'
and is_human
and (
logical_category_id <> -1 or vertical_id = -1
)
```

! В сложных предикатах лучше расставлять скобки для ясности понимания – даже если они и не нужны.

Пример не совсем простого предиката из реального проекта.

WHERE

Предикат в предложении WHERE выполняет реляционную операцию ограничения.

Строки, появляющиеся на выходе предложения FROM, ограничиваются теми, для которых предикат дает значение TRUE.

- p1 AND p2 – пересечение

- $p1 \text{ OR } p2$ – объединение
- $\text{NOT } p1$ – разность

Пример

Нужно написать запрос выбора имен студентов группы M8O-102A-19 без дублей.

```
SELECT DISTINCT name
FROM students
WHERE group_code = 'M8O-102A-19';
```

BETWEEN

Предикат BETWEEN проверяет, попадают ли значения проверяемого выражения в диапазон, задаваемый пограничными выражениями, соединяемыми служебным словом AND.

Как и для предиката сравнения, выражения в предикате BETWEEN должны быть совместимы по типам.

```
<Проверяемое выражение> [NOT] BETWEEN <Начальное
выражение> AND <Конечное выражение>
```

- $p1 \text{ BETWEEN } p2 \text{ AND } p3$ эквивалент $p1 \geq p2 \text{ AND } p1 \leq p3$
- $p1 \text{ NOT BETWEEN } p2 \text{ AND } p3$ эквивалент $\text{NOT } (p1 \text{ BETWEEN } p2 \text{ AND } p3)$

Если значение предиката $\text{exp1 BETWEEN exp2 AND exp3}$ равно TRUE, в общем случае это не означает, что значение предиката $\text{exp1 BETWEEN exp3 AND exp2}$ тоже будет TRUE.

Пример

Нужно написать запрос выбора имен студентов, родившихся с 2001 по 2005 года.

```
select distinct name from students where birthday between '2001-01-01' and '2005-12-31';
```

Предикат IN

Предикат IN определяет, будет ли значение проверяемого выражения обнаружено в наборе значений, который либо явно определен, либо получен с помощью табличного подзапроса.

```
<Проверяемое выражение> [NOT] IN <выражение для  
вычисления значения>
```

Здесь табличный подзапрос – это обычный оператор SELECT, который создает одну или несколько строк для одного столбца, совместимого по типу данных со значением проверяемого выражения.

- Если целевой объект эквивалентен хотя бы одному из указанных в предложении IN значений, истинностное значение предиката IN будет равно TRUE.
- Если для каждого значения X в предложении IN целевой объект <> X, истинностное значение будет равно FALSE.
- Если подзапрос выполняется и результат не содержит ни одной строки (пустая таблица), предикат принимает значение FALSE.

Когда не соблюдается ни одно из упомянутых выше условий, значение предиката равно UNKNOWN.

Пример

Нужно найти студентов, обучающихся в группе M8O-101A-19 или M8O-104A-19 с использованием предиката IN.

```
SELECT DISTINCT name  
FROM students  
WHERE group_code IN ('M8O-101A-19', 'M8O-104A-19');
```

Предикат LIKE

Предикат LIKE сравнивает строку, указанную в первом выражении (оно называется проверяемым значением), с образцом, который определен во втором выражении для вычисления значения строки.

```
<Выражение>  
[NOT] LIKE <Выражение для вычисления значения строки>  
[ESCAPE <символ>]
```

Разрешается использовать два трафаретных символа:

- символ подчеркивания (), который можно применять вместо любого единичного символа в проверяемом значении
- символ процента (%) заменяет последовательность любых символов (число символов в последовательности может быть от 0 и более) в проверяемом значении

Если проверяемое значение соответствует образцу с учетом трафаретных символов, то значение предиката равно TRUE.

👉 Если искомая строка содержит трафаретный символ, то следует задать управляющий символ в предложении ESCAPE. Этот управляющий символ должен использоваться в образце перед трафаретным символом, сообщая о том, что последний следует трактовать как обычный символ.

Например, если в некотором поле следует отыскать все значения, содержащие символ « », то шаблон '%_%' приведет к тому, что будут возвращены все записи из таблицы.

В данном случае шаблон следует записать следующим образом:

```
'%#_%' ESCAPE '#'
```

Для проверки значения на соответствие строке «25%» можно воспользоваться таким предикатом:

```
LIKE '25|%' ESCAPE '|'
```

📌 Истинностное значение предиката LIKE присваивается в соответствии со следующими правилами:

- если либо проверяемое значение, либо образец, либо управляющий символ есть NULL, истинностное значение равно UNKNOWN;
- в противном случае, если проверяемое значение и образец имеют нулевую длину, истинностное значение равно TRUE;

- в противном случае, если проверяемое значение соответствует шаблону, то предикат LIKE равен TRUE;
- если не соблюдается ни одно из перечисленных выше условий, предикат LIKE равен FALSE.

NULL в условиях поиска

 ОШИБКА: WHERE <поле> = NULL

Этому предикату не соответствует ни одной строки, поэтому результирующий набор записей будет пуст, даже если существуют значения NULL.

 IS [NOT] NULL

Позволяет проверить отсутствие (наличие) значения в полях таблицы.

Использование в этих случаях обычных предикатов сравнения может привести к неверным результатам, так как сравнение со значением NULL дает результат UNKNOWN (неизвестно).

Сравнение с NULL-значением согласно предикату сравнения оценивается как UNKNOWN. Строка попадает в результирующий набор только в том случае, если предикат в предложении WHERE есть TRUE.