
# Работа с СУБД SQL

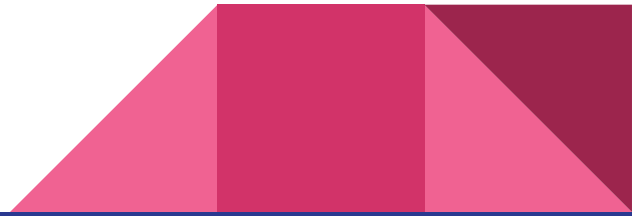
## Common Table Expressions (CTE).


**Проверить, идет ли запись**

# Меня хорошо видно && слышно?

# Проверка

- настройка микрофона и аудио
- проверка работы чата




# Условные операторы

# Оператор CASE

```
CASE
  WHEN условие THEN результат
  [ WHEN ... THEN ... ]
  [ ELSE результат ]
END

CASE выражение
  WHEN значение THEN результат
  [ WHEN ... THEN ... ]
  [ ELSE результат ]
END
```

- аналог цепочки
  **if(...) ... else if(...) ... else if(...) ... else ...,**
  когда на каждом шаге вычисляется значение нового выражения
- **if(...) ... elsif(...) ... elsif(...) ... else ...,**

https://www.db-fiddle.com/f/yfbCAghv9YKLP34J1yHRm/0

# Оператор CASE

```
SELECT
  i
, CASE i % 2
    WHEN 0 THEN 'even'
    WHEN 1 THEN 'odd'
  END v1
, CASE
    WHEN i % 15 = 0 THEN 'foobar'
    WHEN i %  3 = 0 THEN 'foo'
    WHEN i %  5 = 0 THEN 'bar'
  END v2
FROM
  generate_series(0, 15) i;
```

| i | v1 | v2 |
|---|---|---|
| integer | text | text |
| 0 | even | foobar |
| 1 | odd | |
| 2 | even | |
| 3 | odd | foo |
| 4 | even | |
| 5 | odd | bar |
| 6 | even | foo |
| 7 | odd | |
| 8 | even | |
| 9 | odd | foo |
| 10 | even | bar |
| 11 | odd | |
| 12 | even | foo |
| 13 | odd | |
| 14 | even | |
| 15 | odd | foobar |

# Оператор coalesce

В отличие от **CASE**, функция **coalesce** просто последовательно вычисляет значения переданных в нее выражений, пока не встретит первый не-NULL'овый результат - его и возвращает.

То есть coalesce можно рассматривать как подобный CASE:

```
CASE
  WHEN valX IS NOT NULL
    THEN valX
  WHEN valY IS NOT NULL
    THEN valY
END
->
coalesce(valX, valY)
```

# Оператор coalesce

```
SELECT
  i
, coalesce(
    CASE i % 2
      WHEN 0 THEN 'even'
    END
  , CASE
      WHEN i % 15 = 0 THEN 'foobar'
      WHEN i %  3 = 0 THEN 'foo'
      WHEN i %  5 = 0 THEN 'bar'
    END
  )
FROM
  generate_series(0, 15) i;
```

```
 i       | coalesce
integer  | text
       0 | even
       1 |
       2 | even
       3 | foo
       4 | even
       5 | bar
       6 | even
       7 |
       8 | even
       9 | foo
      10 | even
      11 |
      12 | even
      13 |
      14 | even
      15 | foobar
```

https://www.postgresql.org/docs/current/functions-conditional.html

# Оператор nullif

```
SELECT
  i
, nullif(
    i % 2
  , i % 3
  )
FROM
  generate_series(0, 15) i;
```

```
i       | nullif
integer | integer
      0 |          -- i % 2 -> 0 == 0 <- i % 3
      1 |
      2 |        0 -- i % 2 -> 0 == 2 <- i % 3
      3 |        1
      4 |        0
      5 |        1
      6 |
      7 |
      8 |        0
      9 |        1
     10 |        0
     11 |        1
     12 |
     13 |
     14 |        0
     15 |        1
```

СМОТРИ В БУДУЩЕЕ. ИНВЕСТИРУЙ В ЗНАНИЯ.

АИС

# СТЕ

# WITH (CTE)

```
[ WITH [ RECURSIVE ] запрос_WITH [, ...] ]
  имя_CTE [ (имя_столбца, ...) ] AS ( -- Common Table Expression
    { SELECT | TABLE | VALUES |
      { INSERT | UPDATE | DELETE } ... RETURNING ...
    } |
    {
      нерекурсивная_часть
    UNION [ ALL | DISTINCT ]
      рекурсивная_часть
    }
  )
```

https://www.postgresql.org/docs/current/queries-with.html

# CTE

## Без CTE

```sql
SELECT pb.book_id,
       pb.title,
       pb.author,
       s.total_sales
FROM (
    SELECT book_id,
           title,
           author
    FROM books
    WHERE rating >= 4.6
) AS pb
JOIN sales s ON pb.book_id = s.book_id
WHERE s.year = 2022
ORDER BY s.total_sales DESC
LIMIT 5;
```

## С CTE

```sql
WITH popular_books AS (
    SELECT book_id,
           title,
           author
    FROM books
    WHERE rating >= 4.6
),
best_sellers AS (
    SELECT pb.book_id,
           pb.title,
           pb.author,
           s.total_sales
    FROM popular_books pb
    JOIN sales s ON pb.book_id = s.book_id
    WHERE s.year = 2022
    ORDER BY s.total_sales DESC
    LIMIT 5
)
SELECT *
FROM best_sellers;
```

# WITH (CTE)

```
WITH f AS (
  TABLE x -- это обращение к реальной таблице
)
, g AS (
  TABLE f -- это уже обращение к сформированной CTE
)
TABLE g;
```

# WITH (CTE)

```
WITH v AS (
  VALUES
    (1, 2)
)
  TABLE v
UNION ALL
  TABLE v;
```

```
column1 | column2
integer | integer
      1 |       2
      1 |       2
```

```
WITH v(x, y) AS (
  SELECT
    1 a
  , 2 b
)
  TABLE v
UNION ALL
  TABLE v;
```

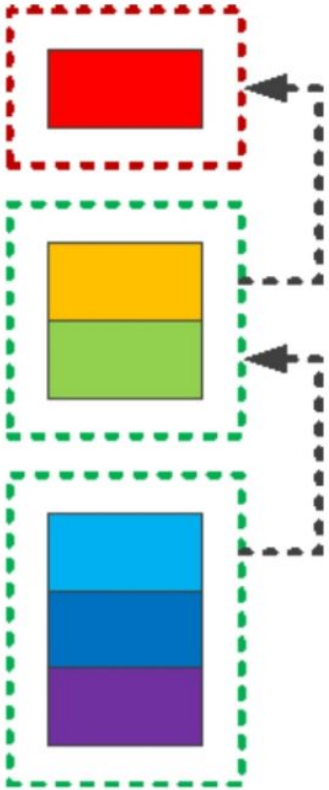
```
x       | y
integer | integer
      1 |       2
      1 |       2
```

# WITH RECURSIVE

```
WITH RECURSIVE fib(i, a, b) AS (
  VALUES(0, 0, 1) -- затравка
UNION ALL
  SELECT          -- шаг рекурсии
    i + 1
  , greatest(a, b)
  , a + b
  FROM
    fib           -- обращение к себе
  WHERE
    i < 10        -- условие продолжения
)
TABLE fib;
```

| i | a | b |
|---|---|---|
| integer | integer | integer |
| 0 | 0 | 1 |
| 1 | 1 | 1 |
| 2 | 1 | 2 |
| 3 | 2 | 3 |
| 4 | 3 | 5 |
| 5 | 5 | 8 |
| 6 | 8 | 13 |
| 7 | 13 | 21 |
| 8 | 21 | 34 |
| 9 | 34 | 55 |
| 10 | 55 | 89 |

# WITH RECURSIVE



На каждом следующем шаге рекурсии такой запрос получает "на вход" (под именем "своей" CTE) результат генерации записей предыдущего сегмента, пока этот результат непустой, или затравочную выборку - для первого шага

Важно понимать, что хоть какое-то условие (по наличию записей, их количеству, счетчику шагов или времени выполнения) должно ограничивать продолжение формирования выборки, иначе есть **риск получить бесконечно выполняющийся запрос.**

# WITH RECURSIVE

```
WITH RECURSIVE exp(i, n) AS (
  VALUES(0, 1)
UNION ALL
  SELECT
    i + 1
  , unnest(ARRAY[n * 2, n * 2])
  FROM
    exp
  WHERE
    i < 2
)
TABLE exp;
```

| i | n |
|---|---|
| integer | integer |
| 0 | 1 |
| 1 | 2 |
| 1 | 2 |
| 2 | 4 |
| 2 | 4 |
| 2 | 4 |
| 2 | 4 |

https://www.db-fiddle.com/f/sEf371RH9xrLcR6e3ntUDq/0

# ДЗ №3

https://github.com/DWH-course-examples/SQL-postgres/blob/main/homework/task3.md

# СПАСИБО ЗА ВНИМАНИЕ!

#аис
#учисьваис