# ***Сделанные задания SQL***

Порядок исполнения команд:



* AND

SELECT title, release\_year

FROM films

WHERE (release\_year > 2000 AND release\_year < 2010)

AND language = 'German';

* OR

SELECT title, release\_year

FROM films

WHERE (release\_year = 1990 OR release\_year = 1999)

    AND (language = 'English' OR language = 'Spanish')

    AND gross > 2000000;

* BETWEEN

SELECT title, release\_year

FROM films

WHERE language = 'Spanish'

    AND release\_year BETWEEN 1990 AND 2000

    AND budget > 100000000

* LIKE / NOT LIKE

SELECT name

FROM people

WHERE name LIKE 'B%'

    OR name LIKE '\_r%'

    AND name NOT LIKE 'A%'

* WHERE, IN

SELECT title, release\_year, certification

FROM films

WHERE year IN(1990, 2000)

    AND certification IN ('NC-17', 'R')

    AND longevity > 120

    AND language IN ('English', 'Spanish', 'French');

* COUNT, DISTINCT, AS

SELECT COUNT(DISTINCT title) AS nineties\_english\_films\_for\_teens

FROM films

WHERE release\_year BETWEEN 1990 AND 1999

    AND language = 'English'

    AND certifications IN ('G', 'PG', 'PG-13');

* NULL

SELECT title AS no\_budget\_info

FROM films

WHERE budget IS NULL;

* NOT NULL

SELECT COUNT(title) AS count\_language\_known

FROM films

WHERE language IS NOT NULL;

* ORDER BY

SELECT release\_year, duration, title

FROM films

ORDER BY release\_year, duration;

* ORDER BY, DESC/ ASC

SELECT release\_year, certification, title

FROM films

ORDER BY certification, release\_year DESC;

## *Aggregate Functions*

* SUM()

SELECT SUM(duration) AS total\_duration

FROM films;

* AVG()

SELECT AVG(duration) AS average\_duration

FROM films;

* MAX()

SELECT MAX(release\_year) AS latest\_year

FROM films;

* MIN()

SELECT MIN(duration) AS shortest\_film

FROM films;

* ROUND()

SELECT ROUND(AVG(facebook\_likes), 1) AS avg\_facebook\_likes

FROM reviews;

SELECT ROUND(AVG(budget), -3) AS avg\_budget\_thousands

FROM films;

## *Grouping Data*

* **GROUP BY**

SELECT release\_year, COUNT(DISTINCT language) AS d\_l

FROM films

GROUP BY release\_year

ORDER BY d\_l DESC;

* **HAVING – use with agg\_func**

SELECT country, AVG(budget) AS average\_budget

FROM films

GROUP BY country

HAVING AVG(budget) > 1000000000

ORDER BY average\_budget DESC;

* **DATE\_FORMAT(column, ‘y%-m%-d%’)**

**SELECT DATE\_FORMAT(shipment\_date, '%Y-%m') date,**

**COUNT(EXTRACT(MONTH FROM shipment\_date)) count\_of\_ship**

**FROM amazon\_shipment**

**GROUP BY date**

***ВСЕ ИЗ КУСРА №2***

SELECT release\_year, AVG(budget) AS avg\_budget, AVG(gross) AS avg\_gross

FROM films

WHERE release\_year > 1990

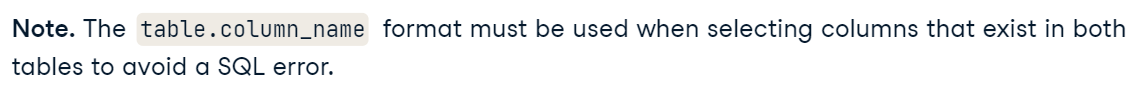
GROUP BY release\_year

HAVING AVG(budget) > 60000000

ORDER BY avg\_gross DESC

LIMIT 1;

## *Joining Data in SQL*



* INNER JOIN

SELECT c.code AS country\_code,

    name,

    year,

    inflation\_rate

FROM countries AS C

INNER JOIN economies AS e

ON c.code = e.code;

* + USING

SELECT c.name AS country, l.name AS language, official

FROM countries AS c

INNER JOIN languages AS l

USING(code);

* Joining multiple tables

SELECT name, e.year, fertility\_rate, e.unemployment\_rate

FROM countries AS c

INNER JOIN populations AS p

ON c.code = p.country\_code

INNER JOIN economies AS e

USING(code)

ORDER BY unemployment\_rate;

/\ - joining table and eliminate double records

SELECT name, e.year, fertility\_rate, unemployment\_rate

FROM countries AS c

INNER JOIN populations AS p

ON c.code = p.country\_code

INNER JOIN economies AS e

ON c.code = e.code

    AND e.year = p.year;

* LEFT JOIN

SELECT region, AVG(gdp\_percapita) AS avg\_gdp

FROM countries AS c

LEFT JOIN economies AS e

USING(code)

WHERE year = 2010

GROUP BY region

ORDER BY AVG(gdp\_percapita) DESC

LIMIT 10;

* RIGHT JOIN

SELECT countries.name AS country, languages.name AS language, percent

FROM languages

RIGHT JOIN countries

USING(code)

ORDER BY language;

* FULL JOIN

SELECT

    c1.name AS country,

    region,

    l.name AS language,

    basic\_unit,

    frac\_unit

FROM countries as c1

FULL JOIN languages as l

USING(code)

FULL JOIN currencies AS c2

USING(code)

WHERE region LIKE 'M%esia';

* **CROSS JOIN** – creates all possible combinations.

SELECT c.name AS country, l.name AS language

FROM countries AS c

CROSS JOIN languages AS l

WHERE c.code in ('PAK','IND')

    AND l.code in ('PAK','IND');

* **SELF JOIN** - Self joins are very useful for comparing data from one part of a table with another part of the same table. Suppose you are interested in finding out how much the populations for each country changed from 2010 to 2015.

SELECT

    p1.country\_code,

    p1.size AS size2010,

    p2.size AS size2015

FROM populations AS p1

INNER JOIN populations AS p2

ON p1.country\_code = p2.country\_code

WHERE p1.year = 2010

    AND p1.year = (p2.year - 5)

### Set Theory for SQL Joins

Both queries on the left and right of the set operation must have the same data types. The names of the fields do not need to be the same, as the result will always contain field names from the left query.

* UNION – without dublicates

SELECT \*

FROM economies2015

UNION

SELECT \*

FROM economies2019

ORDER BY code, year;

* ALL UNION – with dublicates

SELECT code, year

FROM economies

UNION ALL

SELECT country\_code, year

FROM populations

ORDER BY code, year;

* INTERSECT

SELECT name

FROM countries

INTERSECT

SELECT name

FROM cities;

* EXCEPT

SELECT name

FROM cities

EXCEPT

SELECT name

FROM countries

ORDER BY name;

### Subquerying with semi joins and anti joins

* Semi join (also exist anti join)

SELECT DISTINCT name

FROM languages

WHERE code IN

    (SELECT code

    FROM countries

    WHERE region = 'Middle East')

ORDER BY name;

* Subqueries inside **WHERE** (can either be from the same table or a different table).

SELECT \*

FROM populations

WHERE life\_expectancy > 1.15 \* (SELECT AVG(life\_expectancy)

  FROM populations

  WHERE year = 2015)

  AND year = 2015;

* Subqueries inside **SELECT**

SELECT

    name AS league,

    ROUND(AVG(home\_goal + m.away\_goal), 2) AS avg\_goals,

    ROUND(AVG(m.home\_goal + m.away\_goal) -

        (SELECT AVG(home\_goal + away\_goal)

         FROM match

         WHERE season = '2013/2014'), 2) AS diff

FROM league AS l

LEFT JOIN match AS m

ON l.country\_id = m.country\_id

WHERE season = '2013/2014'

GROUP BY l.name;

**/\ - WITHOUT GROUP BY**

SELECT name AS country,

    (SELECT COUNT(\*)

    FROM cities

    WHERE countries.code = cities.country\_code) AS cities\_num

FROM countries

ORDER BY cities\_num DESC,

            country ASC

LIMIT 9;

* Subqueries inside **FROM**

SELECT

    country,

    date,

    home\_goal,

    away\_goal

FROM

    (SELECT name AS country,

            m.date,

            m.home\_goal,

            m.away\_goal,

           (m.home\_goal + m.away\_goal) AS total\_goals

    FROM match AS m

    LEFT JOIN country AS c

    ON m.country\_id = c.id) AS subq

WHERE total\_goals >= 10;

**Все выше:**

SELECT name,

country\_code,

city\_proper\_pop,

metroarea\_pop,

(city\_proper\_pop / metroarea\_pop \* 100) AS city\_perc

FROM cities

WHERE name IN (SELECT capital

        FROM countries

        WHERE (continent = 'Europe' OR continent LIKE ('%America')))

   AND metroarea\_pop IS NOT NULL

ORDER BY city\_perc DESC

LIMIT 10;

**/\**

SELECT

    stage,

    ROUND(s.avg\_goals, 2) AS avg\_goal,

    (SELECT AVG(home\_goal + away\_goal)

    FROM match

    WHERE season = '2012/2013') AS overall\_avg

FROM

    (SELECT

         stage,

         AVG(home\_goal + away\_goal) AS avg\_goals

     FROM match

     WHERE season = '2012/2013'

     GROUP BY stage) AS s

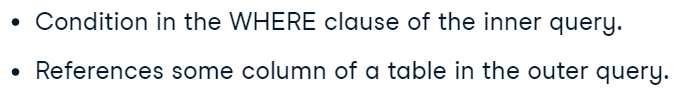
WHERE

    s.avg\_goals > (SELECT AVG(home\_goal + away\_goal)

                    FROM match

                    WHERE season = '2012/2013');

* Correlated Subqueries:



SELECT

    main.country\_id,

    main.date,

    main.home\_goal,

    main.away\_goal

FROM match AS main

WHERE

    (main.home\_goal + main.away\_goal) =

        (SELECT MAX(sub.home\_goal + sub.away\_goal)

         FROM match AS sub

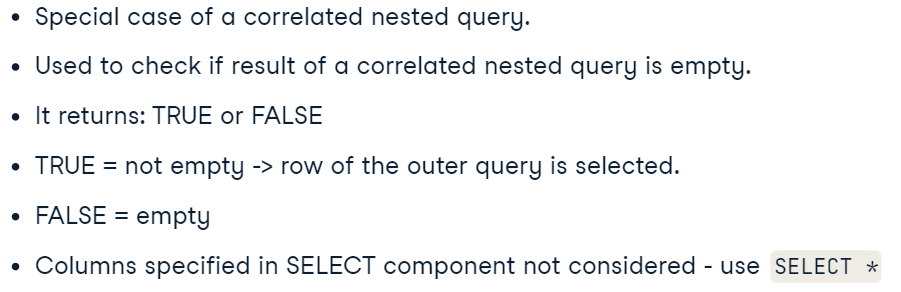
         WHERE sub.season = main.season

               AND main.country\_id = sub.country\_id);

**MORE EXAMPLES**:

<https://campus.datacamp.com/courses/data-driven-decision-making-in-sql/data-driven-decision-making-with-advanced-sql-queries?ex=5>

* EXISTS – works as correlated subquery



SELECT a.nationality,

    COUNT(\*)

FROM actors AS a

WHERE EXISTS

    (SELECT ai.actor\_id

     FROM actsin AS ai

     LEFT JOIN movies AS m

      ON m.movie\_id = ai.movie\_id

     WHERE m.genre = 'Comedy'

        AND ai.actor\_id = a.actor\_id)

GROUP BY a.nationality;

## *Data Manipultion With SQL*

### CASE

* Case

SELECT date,

        CASE WHEN hometeam\_id = 8634 THEN 'FC Barcelona'

            ELSE 'Real Madrid CF' END as home,

        CASE WHEN awayteam\_id = 8634 THEN 'FC Barcelona'

            ELSE 'Real Madrid CF' END as away,

        CASE WHEN home\_goal > away\_goal AND hometeam\_id = 8634 THEN 'Barcelona win!'

          WHEN home\_goal > away\_goal AND hometeam\_id = 8633 THEN 'Real Madrid win!'

          WHEN home\_goal < away\_goal AND awayteam\_id = 8634 THEN 'Barcelona win!'

          WHEN home\_goal < away\_goal AND awayteam\_id = 8633 THEN 'Real Madrid win!'

            ELSE 'Tie!' END AS outcome

FROM matches\_spain

WHERE (awayteam\_id = 8634 OR hometeam\_id = 8634)

      AND (awayteam\_id = 8633 OR hometeam\_id = 8633);

* Agg func with CASE

SELECT

    c.name AS country,

    ROUND(AVG(CASE WHEN m.season='2013/2014' AND m.home\_goal = m.away\_goal THEN 1

             WHEN m.season='2013/2014' AND m.home\_goal != m.away\_goal THEN 0

             END), 2) AS pct\_ties\_2013\_2014,

    ROUND(AVG(CASE WHEN m.season='2014/2015' AND m.home\_goal = m.away\_goal THEN 1

             WHEN m.season='2014/2015' AND m.home\_goal != m.away\_goal THEN 0

             END), 2) AS pct\_ties\_2014\_2015

FROM country AS c

LEFT JOIN matches AS m

ON c.id = m.country\_id

GROUP BY country;

### Nested subqueries

* EXTRACT()

SELECT

    season,

    MAX(home\_goal + away\_goal) AS max\_goals,

   (SELECT MAX(home\_goal + away\_goal) FROM match) AS overall\_max\_goals,

   (SELECT MAX(home\_goal + away\_goal)

    FROM match

    WHERE id IN (

          SELECT id

          FROM match

          WHERE EXTRACT(MONTH FROM date) = 07)) AS july\_max\_goals

FROM match

GROUP BY season;

* Nested subqueries

SELECT

  c.name AS country,

  AVG(outer\_s.matches) AS avg\_seasonal\_high\_scores

FROM country AS c

LEFT JOIN

  (SELECT

      country\_id,

      season,

      COUNT(id) AS matches

  FROM

    (SELECT

      country\_id,

      season,

      id

    FROM match

    WHERE home\_goal >= 5 OR away\_goal >= 5) AS inner\_s

    GROUP BY country\_id, season) AS outer\_s

ON c.id = outer\_s.country\_id

GROUP BY country;

### CTE

* CTE

WITH match\_list AS (

    SELECT

        country\_id,

        (home\_goal+away\_goal) AS goals

    FROM match

    WHERE id IN (

        SELECT id

        FROM match

        WHERE season = '2013/2014' AND

            EXTRACT(MONTH FROM date) = 8

    )

)

SELECT

    l.name,

    AVG(goals)

FROM league AS l

LEFT JOIN match\_list

ON match\_list.country\_id = l.country\_id

GROUP BY l.name

* Correlated subqueries

SELECT

    main.country\_id,

    main.date,

    main.home\_goal,

    main.away\_goal

FROM match AS main

WHERE

    (main.home\_goal + main.away\_goal) =

        (SELECT MAX(sub.home\_goal + sub.away\_goal)

         FROM match AS sub

         WHERE sub.season = main.season

               AND main.country\_id = sub.country\_id);

/\

SELECT

    m.date,

   (SELECT team\_long\_name

   FROM team AS t

   WHERE t.team\_api\_id=m.hometeam\_id) AS hometeam,

   (SELECT team\_long\_name

   FROM team AS t

   WHERE t.team\_api\_id=m.awayteam\_id) AS awayteam,

   home\_goal,

   away\_goal

FROM match AS m

* + **ALL KNOWLEDGE**

WITH home AS (

  SELECT m.id, t.team\_long\_name,

      CASE WHEN m.home\_goal > m.away\_goal THEN 'MU Win'

           WHEN m.home\_goal < m.away\_goal THEN 'MU Loss'

           ELSE 'Tie' END AS outcome

  FROM match AS m

  LEFT JOIN team AS t

    ON m.hometeam\_id = t.team\_api\_id),

away AS (

  SELECT m.id, t.team\_long\_name,

    CASE WHEN m.home\_goal > m.away\_goal THEN 'MU Loss'

        WHEN m.home\_goal < m.away\_goal THEN 'MU Win'

            ELSE 'Tie' END AS outcome

FROM match AS m

LEFT JOIN team AS t

ON m.awayteam\_id = t.team\_api\_id)

SELECT DISTINCT m.date,

    home.team\_long\_name AS home\_team,

    away.team\_long\_name AS away\_team,

    m.home\_goal,

    m.away\_goal,

    RANK() OVER(ORDER BY ABS(home\_goal-away\_goal) DESC) as match\_rank

FROM match AS m

LEFT JOIN home ON m.id = home.id

LEFT JOIN away ON m.id = away.id

WHERE season='2014/2015'

    AND ((home.team\_long\_name='Manchester United'

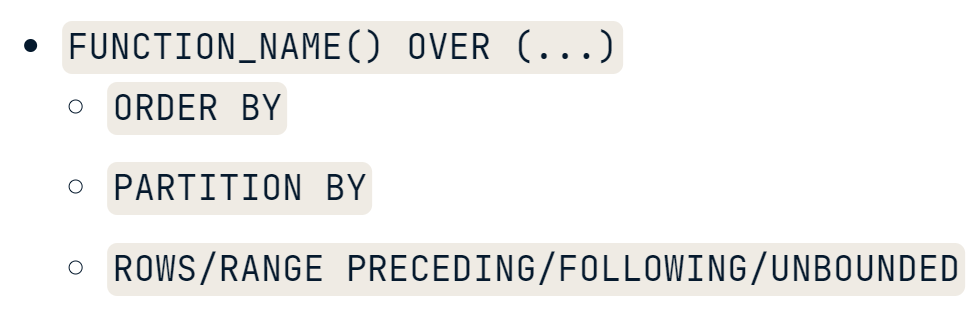
            AND home.outcome = 'MU Loss')

    OR (away.team\_long\_name='Manchester United'

            AND away.outcome = 'MU Loss'))

## Window Functions

### Over, Partition by



* OVER()

SELECT

    m.id,

    c.name AS country,

    m.season,

    m.home\_goal,

    m.away\_goal,

    AVG(m.home\_goal + m.away\_goal) OVER() AS overall\_avg

FROM match AS m

LEFT JOIN country AS c

ON m.country\_id = c.id;

* OVER(ORDER BY) – operate as a subquery

WITH Athlete\_Medals AS (

  SELECT

    Athlete,

    COUNT(\*) AS Medals

  FROM Summer\_Medals

  GROUP BY Athlete)

SELECT

  Athlete,

  ROW\_NUMBER() OVER (ORDER BY Medals DESC) AS Row\_N

FROM Athlete\_Medals

ORDER BY Medals DESC;

* PARTITION BY with OVER() – separetes calculating by different levels.

*(For example, you can create a single column that calculates an overall average of goals scored for each month in season.)*

SELECT

    date,

    season,

    home\_goal,

    away\_goal,

    CASE WHEN hometeam\_id = 8673 THEN 'home'

         ELSE 'away' END AS warsaw\_location,

    AVG(home\_goal) OVER(PARTITION BY season,

            EXTRACT(MONTH FROM date)) AS season\_mo\_home,

    AVG(away\_goal) OVER(PARTITION BY season,

            EXTRACT(MONTH FROM date)) AS season\_mo\_away

FROM match

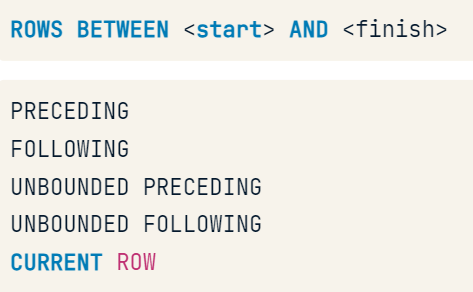
WHERE

    hometeam\_id = 8673

    OR awayteam\_id = 8673

ORDER BY (home\_goal + away\_goal) DESC;

### Sliding windows – (BETWEEN … AND …)



**RANGE** treats duplicates in the columns in the ORDER BY subclause as single entities, whereas **ROWS** does not.

**Sliding windows allow you to create running calculations between any two points in a window using functions such as PRECEDING, FOLLOWING, and CURRENT ROW.**

* UNBOUNDED PRECEDING

SELECT

     date,

     home\_goal,

     away\_goal,

    SUM(home\_goal) OVER(ORDER BY date

         ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS running\_total,

    AVG(home\_goal) OVER(ORDER BY date

         ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS running\_avg

FROM match

WHERE

     hometeam\_id = 9908

     AND season = '2011/2012';

* UNBOUNDED FOLLOWING

SELECT

     date ,

     home\_goal,

     away\_goal,

     SUM(home\_goal) OVER(ORDER BY date DESC

         ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) AS running\_total,

     AVG(home\_goal) OVER(ORDER BY date DESC

     ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) AS running\_avg

FROM match

WHERE

     awayteam\_id = 9908

    AND season = '2011/2012';

* BETWEEN CURRENT ROW N FOLLOWING

SELECT

  Year,

  Medals,

  MAX(Medals) OVER (ORDER BY YEAR ASC

             ROWS BETWEEN CURRENT ROW

             AND 1 FOLLOWING) AS Max\_Medals

FROM Scandinavian\_Medals

ORDER BY Year ASC;

* BETWEEN N PRECEDING AND CURRENT ROW

SELECT

  Athlete,

  Medals,

  MAX(Medals) OVER (ORDER BY Athlete ASC

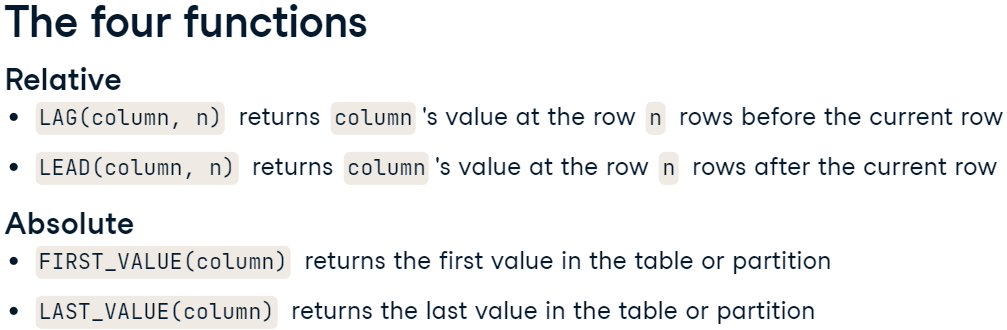
            ROWS BETWEEN 2 PRECEDING

            AND CURRENT ROW) AS Max\_Medals

FROM Chinese\_Medals

ORDER BY Athlete ASC;

### Fetching (LAG, LEAD)



* LAG(column, n) OVER() - takes a column and a number ‘n’ and returns the column's value ‘n’ rows before the current row.

WITH Weightlifting\_Gold AS (

  SELECT

    Year,

    Country AS champion

  FROM Summer\_Medals

  WHERE

    Discipline = 'Weightlifting' AND

    Event = '69KG' AND

    Gender = 'Men' AND

    Medal = 'Gold')

SELECT

  Year, Champion,

  LAG(Champion) OVER

    (ORDER BY Year ASC) AS Last\_Champion

FROM Weightlifting\_Gold

ORDER BY Year ASC;

* Patition by with LAG() – helps not to include irrelative data.

WITH Athletics\_Gold AS (

  SELECT DISTINCT

    Gender, Year, Event, Country

  FROM Summer\_Medals

  WHERE

    Year >= 2000 AND

    Discipline = 'Athletics' AND

    Event IN ('100M', '10000M') AND

    Medal = 'Gold')

SELECT

  Gender,

  Year,

  Event,

  Country AS Champion,

  LAG(Country)

      OVER(PARTITION BY gender, event ORDER BY year) AS Last\_Champion

FROM Athletics\_Gold

ORDER BY gender

You can partition by more than one column in case your groups are spread over several columns.

* Lead(column, int) OVER() – there is a photo higher.

WITH Discus\_Medalists AS (

  SELECT DISTINCT Year,

      Athlete

  FROM Summer\_Medals

  WHERE Medal = 'Gold'

    AND Event = 'Discus Throw'

    AND Gender = 'Women'

    AND Year >= 2000)

SELECT

  year,

  athlete,

  LEAD(athlete, 3) OVER (ORDER BY year ASC) AS Future\_Champion

FROM Discus\_Medalists

ORDER BY Year ASC;

* FIRST\_VALUE() – to compare the first value to the current one.

SELECT

  athlete,

  FIRST\_VALUE(athlete) OVER(ORDER BY athlete) AS First\_Athlete

FROM All\_Male\_Medalists;

* LAST\_VALUE() – to compare the last value to the current one.

SELECT

  Year, City,

  LAST\_VALUE(City) OVER (

   ORDER BY Year

   RANGE BETWEEN

     UNBOUNDED PRECEDING AND

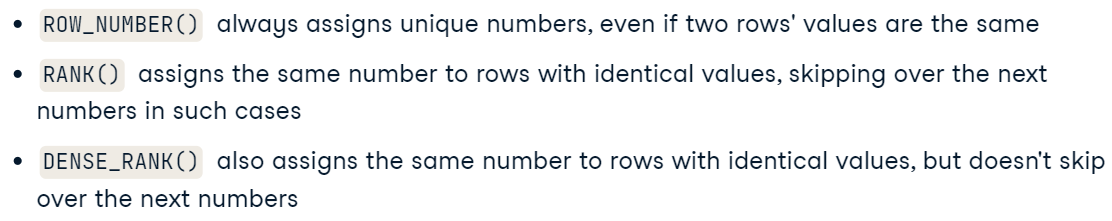
     UNBOUNDED FOLLOWING

  ) AS Last\_City

FROM Hosts

ORDER BY Year ASC;

### Ranking function



* ROW\_NUMBER() – assigns a row number to it’s row.

SELECT \*,

  ROW\_NUMBER() OVER() AS Row\_N

FROM Summer\_Medals

ORDER BY Row\_N ASC;

* RANK() - there is a photo higher.

SELECT

    l.name AS league,

    AVG(m.home\_goal + m.away\_goal) AS avg\_goals,

    RANK() OVER(ORDER BY AVG(m.home\_goal + m.away\_goal)) AS league\_rank

FROM league AS l

LEFT JOIN match AS m

ON l.id = m.country\_id

WHERE m.season = '2011/2012'

GROUP BY l.name

ORDER BY league\_rank;

* DENSE\_RANK() – there is a photo higher.

SELECT

  Country,

  Athlete,

  DENSE\_RANK() OVER (PARTITION BY Country

                ORDER BY Medals DESC) AS Rank\_N

FROM Athlete\_Medals

ORDER BY Country ASC, RANK\_N ASC;

### Agg func in window func

* Cumsum

SELECT

  athlete,

  Medals,

  SUM(Medals) OVER (ORDER BY athlete ASC) AS Max\_Medals

FROM Athlete\_Medals

ORDER BY Athlete ASC;

* (cummax / cummin) – comparing values between current row and (max/min) value.

SELECT

Year,

Country,

Medals,

MAX(Medals) OVER (PARTITION BY Country

ORDER BY Year ASC) AS Max\_Medals

FROM Country\_Medals

ORDER BY Country ASC, Year ASC;

* Moving average of …

Using frames with aggregate window functions allow you to calculate many common metrics, including moving averages and totals. These metrics track the change in performance over time.

SELECT

  Year, Medals,

  AVG(Medals) OVER

    (ORDER BY Year ASC

     ROWS BETWEEN

     2 PRECEDING AND CURRENT ROW) AS Medals\_MA

FROM Russian\_Medals

ORDER BY Year ASC;

* Moving total of …

SELECT

  Year,

  Country,

  Medals,

  SUM(Medals) OVER

    (PARTITION BY Country

     ORDER BY Year ASC

     ROWS BETWEEN

     2 PRECEDING AND CURRENT ROW) AS Medals\_MA

FROM Country\_Medals

ORDER BY Country ASC, Year ASC;

### ROLLUP AND CUBE

* ROLLUP() - is a GROUP BY subclause that includes extra rows for group-level (sum) aggregations.

SELECT

  country,

  gender,

  COUNT(\*) AS Gold\_Awards

FROM Summer\_Medals

WHERE

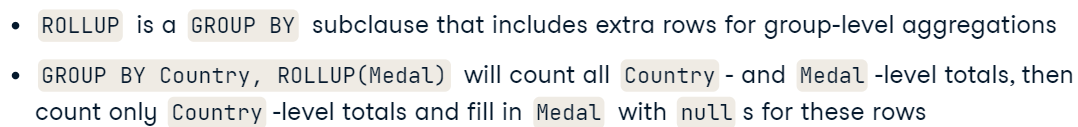
  Year = 2004

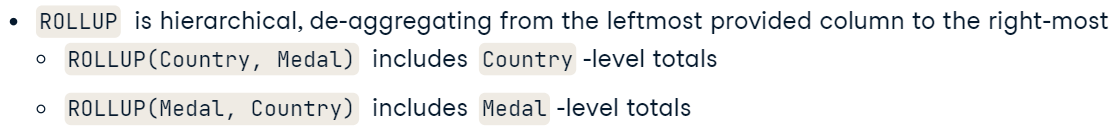
  AND Medal = 'Gold'

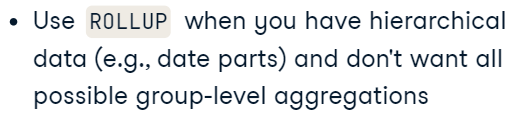
  AND Country IN ('DEN', 'NOR', 'SWE')

GROUP BY country, ROLLUP(gender)

ORDER BY Country ASC, Gender ASC;







* CUBE() - generates all possible group-level aggregations.

SELECT

  Gender,

  Medal,

  COUNT(\*) AS Awards

FROM Summer\_Medals

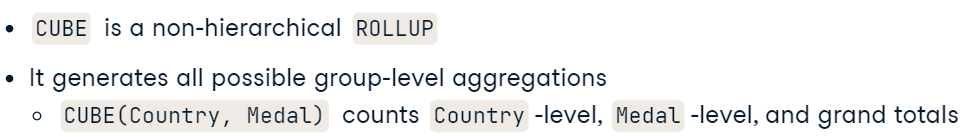
WHERE

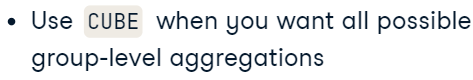
  Year = 2012

  AND Country = 'RUS'

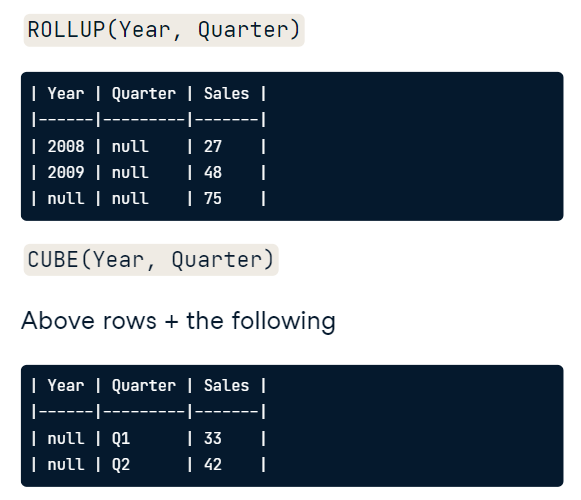
GROUP BY CUBE(Gender, Medal)

ORDER BY Gender ASC, Medal ASC;

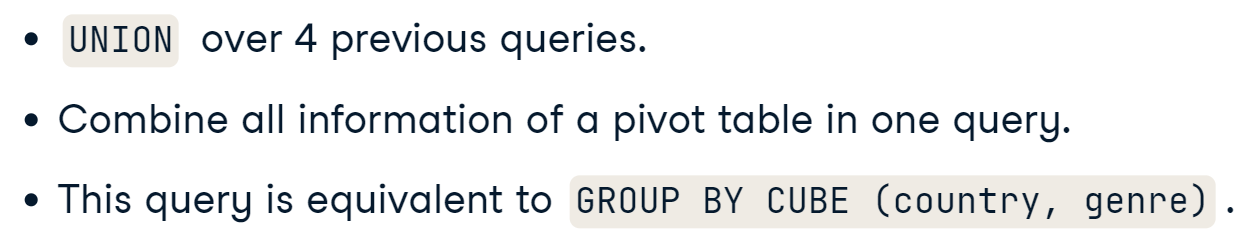


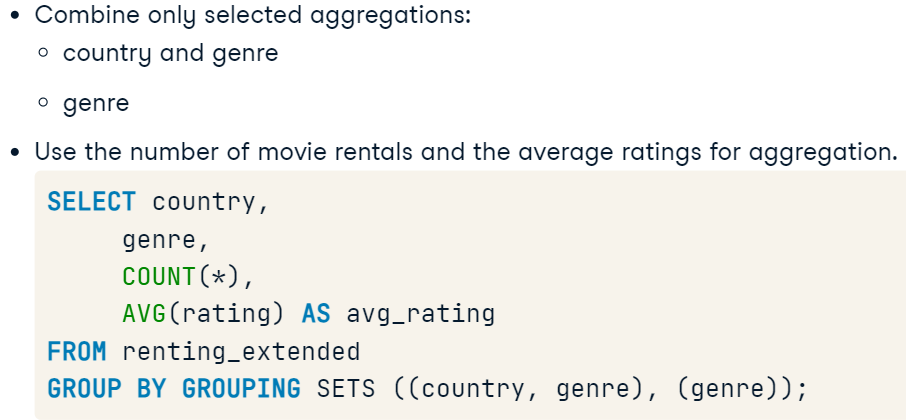


Difference:

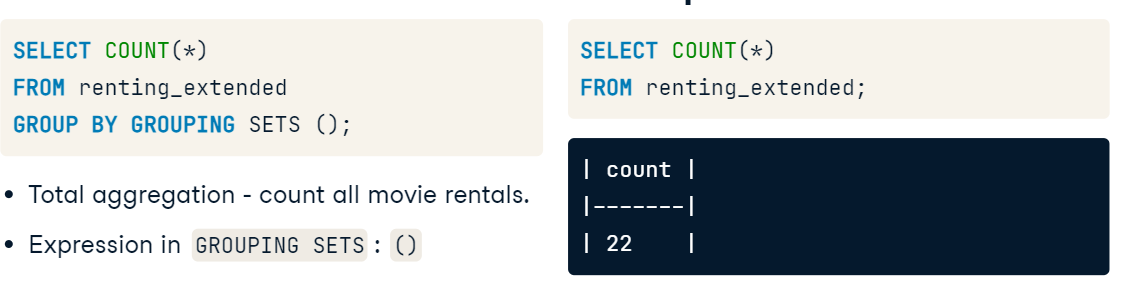


* GROUPING SETS – union all the predetermined tables and conduct total agg with “()” in the end of the query.  **May group by every aspect.**





* While group by **empty par**  – gives the total (NULL, NULL):



* Example of the code:

SELECT a.nationality,

       a.gender,

       AVG(r.rating) AS avg\_rating,

       COUNT(r.rating) AS n\_rating,

       COUNT(\*) AS n\_rentals,

       COUNT(DISTINCT a.actor\_id) AS n\_actors

FROM renting AS r

LEFT JOIN actsin AS ai

ON ai.movie\_id = r.movie\_id

LEFT JOIN actors AS a

ON ai.actor\_id = a.actor\_id

WHERE r.movie\_id IN (

    SELECT movie\_id

    FROM renting

    GROUP BY movie\_id

    HAVING COUNT(rating) >= 4)

AND r.date\_renting >= '2018-04-01'

GROUP BY GROUPING SETS ((a.nationality, a.gender),

                        (a.nationality),

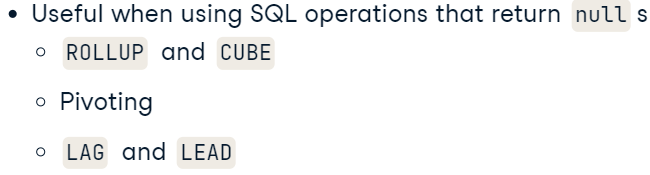
                        (a.gender),

                        ())

ORDER BY AVG(r.rating) DESC

### COALESCE and STRING\_AGG

* COALESCE(column, ‘name’) - takes a list of values and returns the first non-null value, going from left to right.



SELECT

  COALESCE(Country, 'All countries') AS Country,

  COALESCE(Gender, 'All genders') AS Gender,

  COUNT(\*) AS Awards

FROM Summer\_Medals

WHERE

  Year = 2004

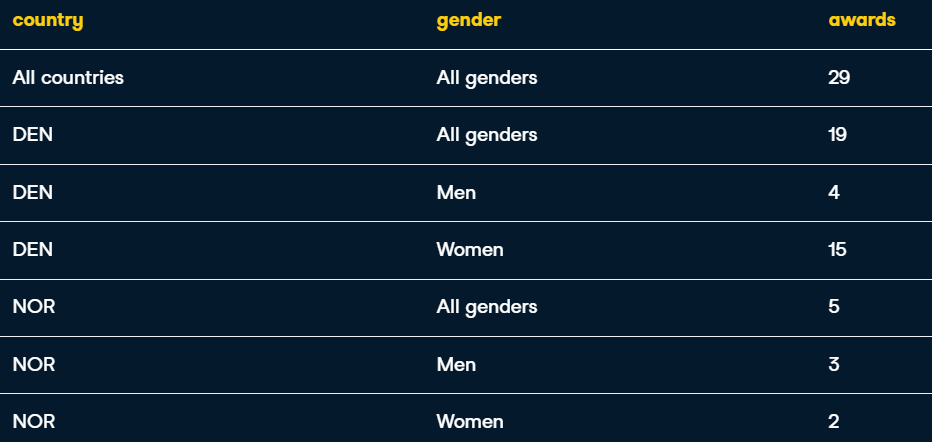
  AND Medal = 'Gold'

  AND Country IN ('DEN', 'NOR', 'SWE')

GROUP BY ROLLUP(Country, Gender)

ORDER BY Country ASC, Gender ASC;

Result:



* STRING\_AGG(column, separator) - takes all the values of a column and concatenates them, with a separator in between each value. Compressing three rows into one, losing no information in the process.

WITH Country\_Medals AS (

  SELECT

    Country,

    COUNT(\*) AS Medals

  FROM Summer\_Medals

  WHERE Year = 2000

    AND Medal = 'Gold'

  GROUP BY Country),

Country\_Ranks AS (

  SELECT

    Country,

    RANK() OVER (ORDER BY Medals DESC) AS Rank

  FROM Country\_Medals

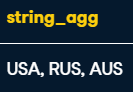
  ORDER BY Rank ASC)

SELECT STRING\_AGG(Country, ', ')

FROM Country\_Ranks

WHERE RANK <= 3;

Result:



### Pivoting

CREATE EXTENSION IF NOT EXISTS tablefunc;

* CROSSTAB

CREATE EXTENSION IF NOT EXISTS tablefunc;

SELECT \* FROM CROSSTAB($$

  WITH Country\_Awards AS (

    SELECT

      Country,

      Year,

      COUNT(\*) AS Awards

    FROM Summer\_Medals

    WHERE

      Country IN ('FRA', 'GBR', 'GER')

      AND Year IN (2004, 2008, 2012)

      AND Medal = 'Gold'

    GROUP BY Country, Year)

  SELECT

    Country,

    Year,

    RANK() OVER(PARTITION BY year

                ORDER BY Awards) :: INTEGER AS RANK

  FROM Country\_Awards

  ORDER BY Country ASC, Year ASC

$$) AS ct (Country VARCHAR,

          "2004-12-01" INTEGER,

          "2008-12-01" INTEGER,

          "2012-12-01" INTEGER)

ORDER BY Country

### Paging

* NTILE(n) – devides df into n pieces.

Splitting your data into thirds or quartiles is often useful to understand how the values in your dataset are spread. Getting summary statistics (averages, sums, standard deviations, etc.) of the top, middle, and bottom thirds can help you determine what distribution your values follow.

Thirds AS (

  SELECT

    Athlete,

    Medals,

    NTILE(3) OVER (ORDER BY Medals DESC) AS Third

  FROM Athlete\_Medals)

SELECT

  Third,

  ROUND(AVG(Medals), 2) AS Avg\_Medals

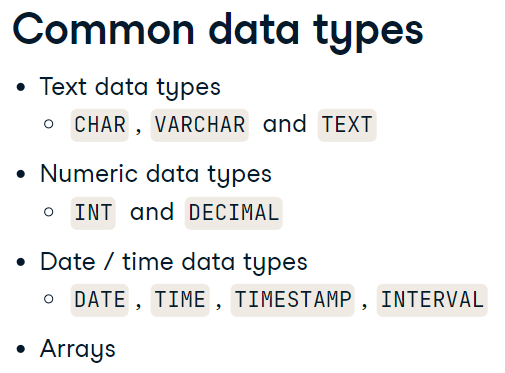
FROM Thirds

GROUP BY Third

ORDER BY Third

## Functions for Manipulating Data

**Common data types:**



* + **INTERVAL types (**representations of periods of time**)** store date and time data as a period of time in years, months, days, hours, seconds, etc. Also, provide you with a very useful tool for performing arithmetic on date and time data types.
  + TIMESTAMP data types contain both date and time values.

### INFORMATION\_SCHEMA

PostgreSQL has a system database called **INFORMATION\_SCHEMA** that allows us to extract information about objects, including tables, in our **WHOLE** database.

* INFORMATION\_SCHEMA.TABLES

SELECT \*

FROM INFORMATION\_SCHEMA.TABLES

WHERE table\_schema = 'public';

Result:

* INFORMATION\_SCHEMA.COLUMNS

SELECT \*

FROM INFORMATION\_SCHEMA.COLUMNS

WHERE table\_name = 'actor';

* Determine type of each column in the table.

SELECT

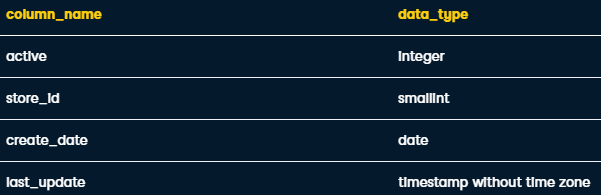
    column\_name,

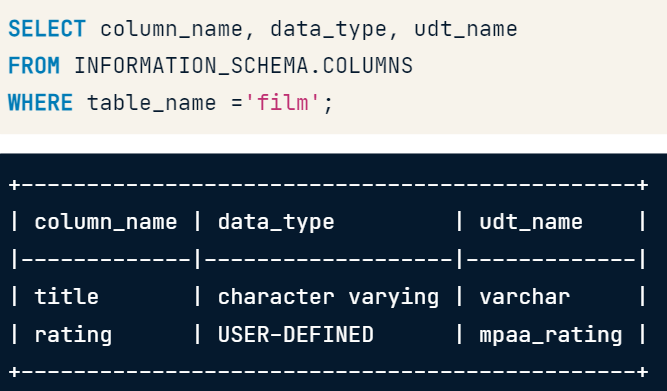
    data\_type

FROM INFORMATION\_SCHEMA.COLUMNS

WHERE table\_name = 'customer';

Result:





### Date and data types

* [**Interval**](#интервал)

SELECT

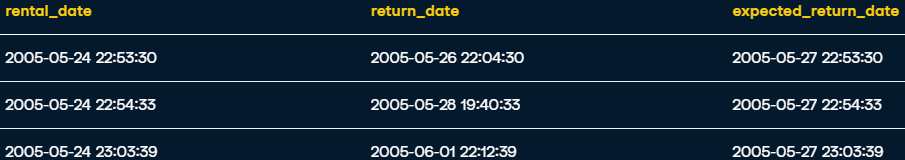
    rental\_date,

    return\_date,

    rental\_date + **INTERVAL** '3 days' as expected\_return\_date

FROM rental

Result:



### Arrays

* Working with arrays

SELECT

  title,

  special\_features

FROM film

WHERE special\_features[2] = 'Deleted Scenes'

* ANY – function allows you to search for a value in any index position of an ARRAY.

SELECT

  title,

  special\_features

FROM film

WHERE 'Trailers' = ANY(special\_features)

/\

The contains operator @> operator is alternative syntax to the ANY function and matches data in an ARRAY using the following syntax.

SELECT

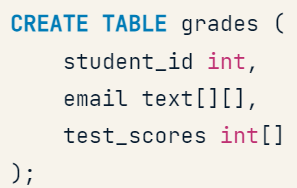
  title,

  special\_features

FROM film

WHERE special\_features @> ARRAY['Deleted Scenes']

* Creating ARRAY-type columns:



* + Inserting values:



### String and character data

#### Reformatting string

* Concat columns:

SELECT

    first\_name || ' ' || last\_name ||' <'|| email || '>' AS full\_email

FROM customer

/\

SELECT

    CONCAT(first\_name, ' ', last\_name, ' <', email, '>') AS full\_email

FROM customer

Result:



* UPPER() / LOWER() / INITCAP()

SELECT

  CONCAT(UPPER(c.name), ': ', INITCAP(f.title)) AS film\_category,

  LOWER(description) AS description

FROM film f

LEFT JOIN film\_category fc

  ON f.film\_id = fc.film\_id

LEFT JOIN category c

  ON fc.category\_id = c.category\_id

* REPLACE(column, str\_db, str\_your) – replacing values

SELECT

  REPLACE(title, ' ', '\_') as title

FROM film

Result:



#### Parsing string

* CHAR\_LENGTH() / LENGTH() – return the len of the string-value.

SELECT

  title,

  description,

  CHAR\_LENGTH(description) AS desc\_len

FROM film

* LEFT() / RIGHT() – extracts value in a sertain amout.

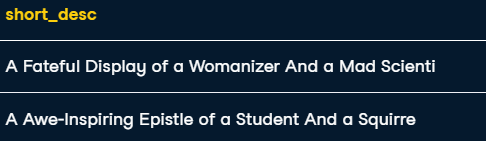
SELECT

  LEFT(description, 50) AS short\_desc

FROM

  film AS f;

result:



* Substring() – extracts strgs from specific position in the str.

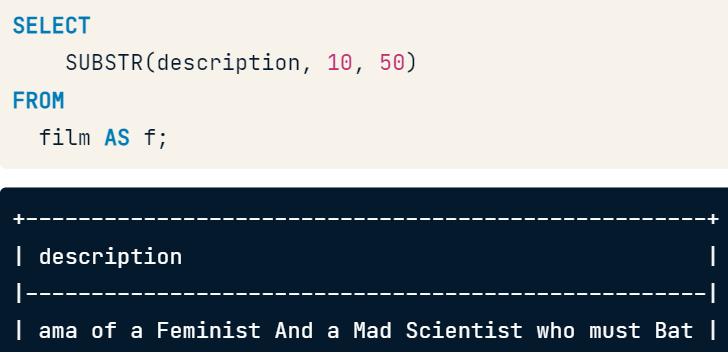
SELECT

  SUBSTRING(address FROM POSITION(' ' in address)+1 FOR CHAR\_LENGTH(address))

FROM

  address;

* + Substr() – an alternative to Substring, but without (FROM … FOR …) clause.



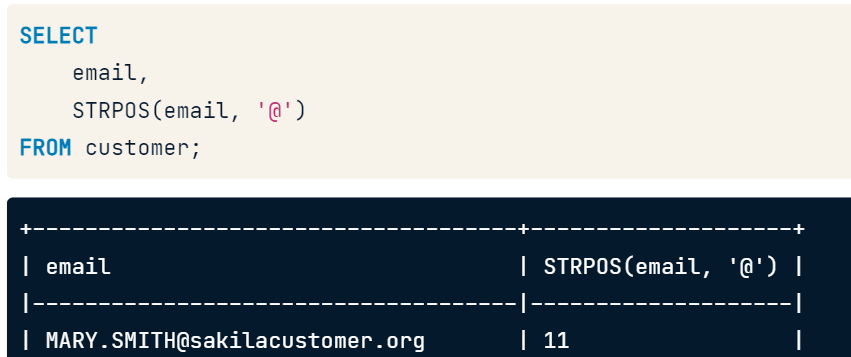
* POSITION() / STRPOS() – extracts the index of the value we search

SELECT

  SUBSTRING(email FROM 0 FOR POSITION('@' IN email)) AS username,

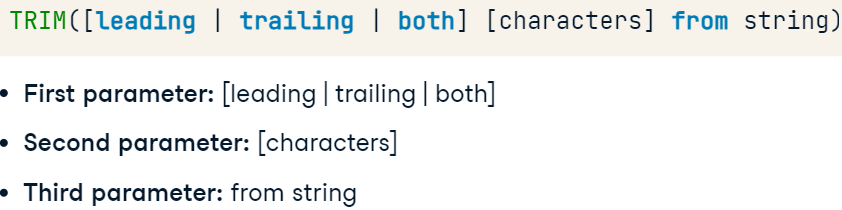
  SUBSTRING(email FROM POSITION('@' IN email)+1 FOR LENGTH(email)) AS domain

FROM customer



#### Truncating and padding string data

* TRIM() / LTRIM() / RTRIM()





SELECT

  CONCAT(UPPER(c.name), ': ', f.title) AS film\_category,

  TRIM(LEFT(f.description, 50)) AS film\_desc

FROM

  film AS f

  INNER JOIN film\_category AS fc

    ON f.film\_id = fc.film\_id

  INNER JOIN category AS c

    ON fc.category\_id = c.category\_id;

/\ - truncating text fields like the film table's description column without cutting off a word.

SELECT

  UPPER(c.name) || ': ' || f.title AS film\_category,

  LEFT(description, 50 -

    POSITION(' ' IN REVERSE(LEFT(description, 50)))

  )

FROM

  film AS f

  INNER JOIN film\_category AS fc

    ON f.film\_id = fc.film\_id

  INNER JOIN category AS c

    ON fc.category\_id = c.category\_id;

* LPAD() / RPAD()



SELECT

    RPAD(first\_name, LENGTH(first\_name)+1, ' ')

    || RPAD(last\_name, LENGTH(last\_name)+2, ' <')

    || RPAD(email, LENGTH(email)+1, '>') AS full\_email

FROM customer

* Unnest(string\_to\_array()) – breaks down string with tons of values into separate rows:

WITH cats AS (

SELECT unnest(string\_to\_array(categories, ';')) AS category,

review\_count

FROM yelp\_business

)

SELECT category,

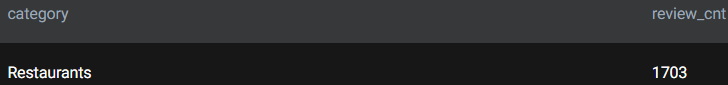
sum(review\_count) as review\_cnt

FROM cats

GROUP BY category

ORDER BY review\_cnt DESC

Result:



### Date manipulation

* Simple math-operations with ***timestamps***:

SELECT f.title, f.rental\_duration,

    r.return\_date - r.rental\_date AS days\_rented

FROM film AS f

    INNER JOIN inventory AS i

        ON i.film\_id = f.film\_id

    INNER JOIN rental AS r

        ON r.inventory\_id = i.inventory\_id

ORDER BY f.title;

/\

SELECT

    f.title,

    INTERVAL '1' day \* f.rental\_duration,

    AGE(r.return\_date, r.rental\_date) AS days\_rented

FROM film AS f

    INNER JOIN inventory AS i ON f.film\_id = i.film\_id

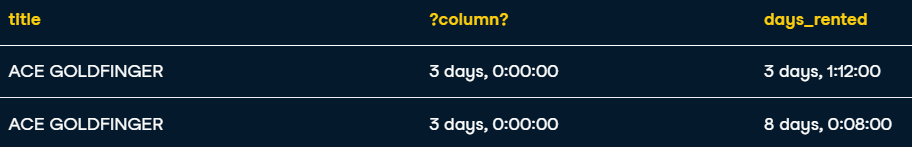
    INNER JOIN rental AS r ON i.inventory\_id = r.inventory\_id

WHERE return\_date IS NOT NULL

ORDER BY f.title;

* + The second arg converts “rental\_duration”(int) into *INTERVAL*.

Result:



* **AGE(date, date)** – subtracts values:

SELECT f.title, f.rental\_duration,

    AGE(r.return\_date, r.rental\_date) AS days\_rented

FROM film AS f

    INNER JOIN inventory AS i ON f.film\_id = i.film\_id

    INNER JOIN rental AS r ON i.inventory\_id = r.inventory\_id

ORDER BY f.title;

#### Functions for retrieving current date/time

* NOW() – returns a timestamp with timezone by default.
* CURRENT\_TIMESTAMP() – returns a timestamp with timezone by default and lets you set up precision of the time.
* CURRENT\_TIME – retrieves current time.
* CURRENT\_DATE – retrieves current date.
* CAST() – transformes column into specific data\_type:

SELECT CAST(NOW() AS timestamp)

#### Extracting and transforming date/ time data

* EXTRACT()

SELECT

  EXTRACT(dow FROM rental\_date) AS dayofweek,

  COUNT(\*) AS rentals

FROM rental

GROUP BY 1;

* + Dow – the day of the week
* DATE\_TRANC() - function will truncate timestamp or interval data types to return a timestamp or interval at a specified precision.

SELECT

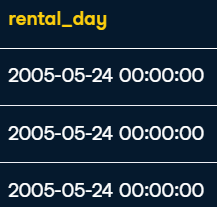
  DATE\_TRUNC('day', rental\_date) AS rental\_day,

  COUNT(\*) AS rentals

FROM rental

GROUP BY 1

Result:



/\

SELECT

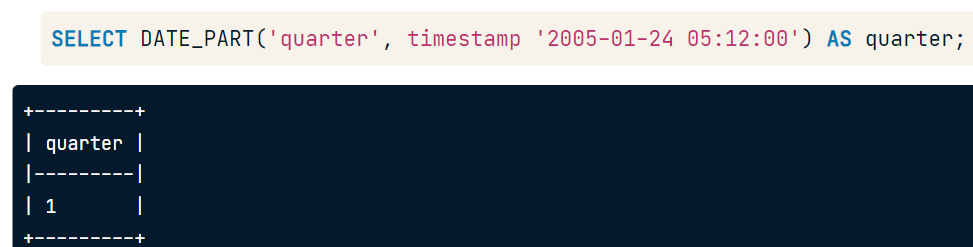
    DATE\_TRUNC('year', rental\_date) AS rental\_year

FROM rental

Result:



* DATE\_PART() as Extract



### Full-text search and Extentions

#### Full-text search

* LIKE – searches value

SELECT \*

FROM film

WHERE title LIKE '%GOLD%';

* *To\_tsvector* and *to\_tsquery*



Result:

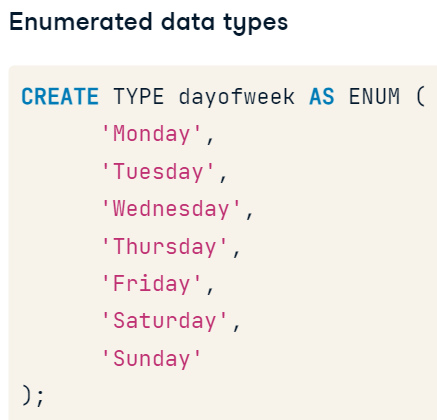
SELECT title, description

FROM film

WHERE to\_tsvector(title) @@ to\_tsquery('elf')

#### Extensions

* **Enumerated data types**:



CREATE TYPE compass\_position AS ENUM (

    'North',

    'South',

    'East',

    'West'

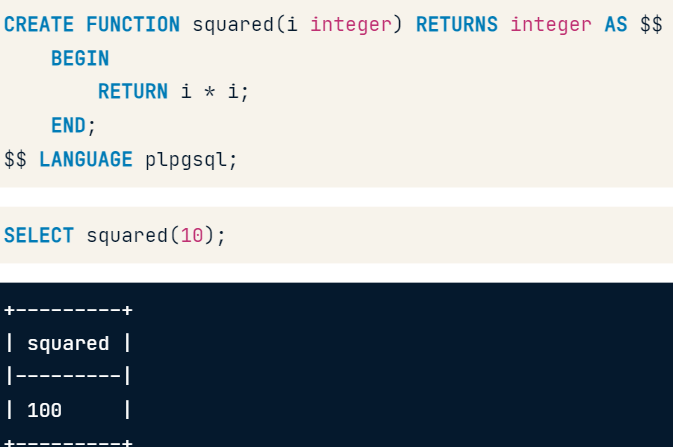
);

SELECT \*

FROM pg\_type

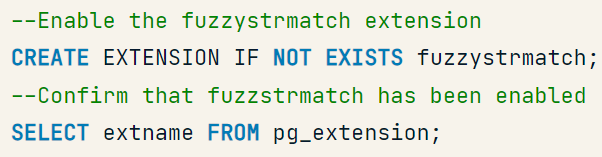
WHERE typname='compass\_position'

#### FUNCTIONS



### PostgreSQL extensions

* Importing extensions:



* Finding available extensions:

SELECT NAME

FROM pg\_available\_extensions

* Finding the similarity between words

SELECT

  title,

  description,

  similarity(description, 'Astounding & Drama')

FROM

  film

WHERE

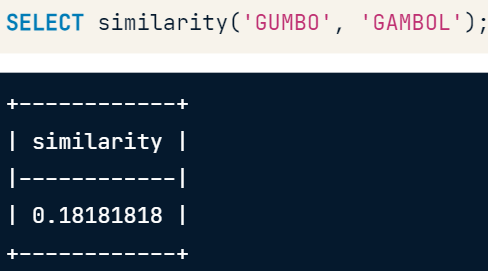
  to\_tsvector(description) @@

  to\_tsquery('Astounding & Drama')

ORDER BY

  similarity(description, 'Astounding & Drama') DESC

Result:



* Finding difference (“distance”) between words:

SELECT

  title,

  description,

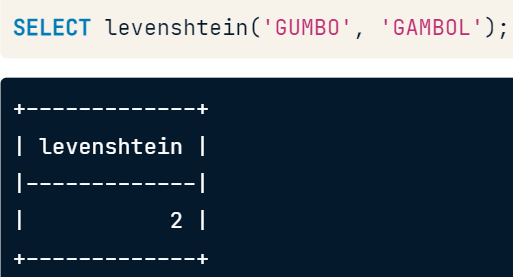
  levenshtein(title, 'JET NEIGHBOR') AS distance

FROM

  film

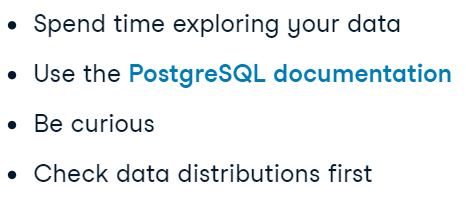
ORDER BY 3

Result:



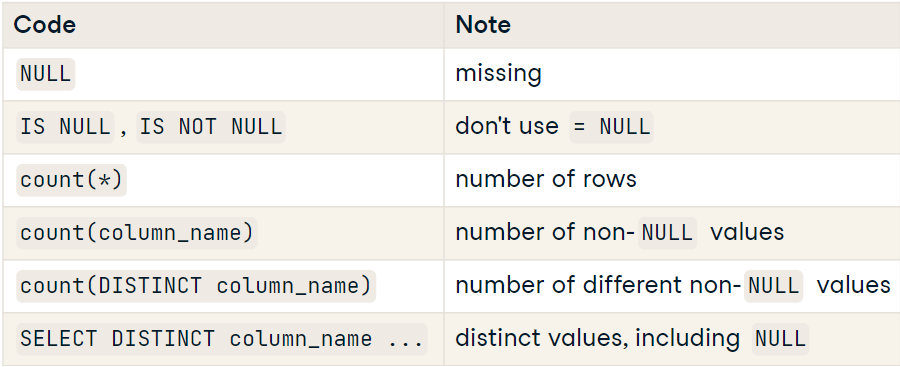
## Exploratory Data Analysis in SQL

#### RULES



#### Necessery basics!!!

* Presentiong the data:



* Rules:

1. Find missing values: count(\*) – count(column)
2. You can join tables when they share a column with consistent data values.
3. Foreign key: reference another row in the database via a unique ID and must reference a column with unique values for each row so the referenced row can be identified. Values in a foreign key column are restricted to values in the referenced column OR NULL.

#### Coalesce

The coalesce() function can be useful for specifying a default or backup value when a column contains NULL values.

coalesce() checks arguments in order and returns the first non-NULL value, if one exists.

* coalesce(NULL, 1, 2) = 1
* coalesce(NULL, NULL) = NULL
* coalesce(2, 3, NULL) = 2
* Example of code:

SELECT coalesce(industry, sector, 'Unknown') AS industry2,

    count(\*)

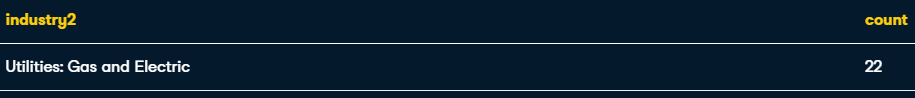
FROM fortune500

GROUP BY industry2

ORDER BY count DESC

limit 1

result:



/\

SELECT company\_original.name,

    title,

    rank

FROM company AS company\_original

LEFT JOIN company AS company\_parent

  ON company\_original.parent\_id = company\_parent.id

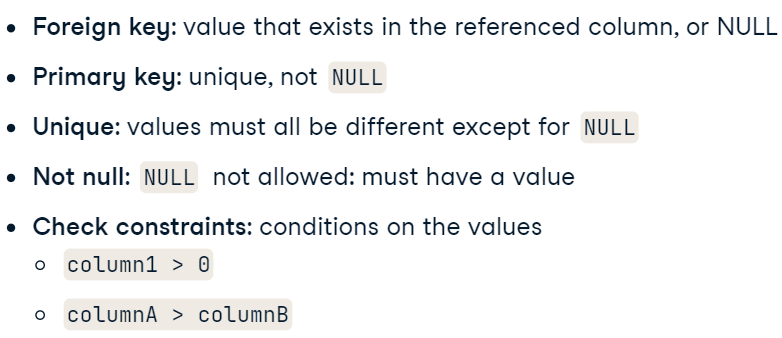
INNER JOIN fortune500

  ON coalesce(company\_original.ticker,

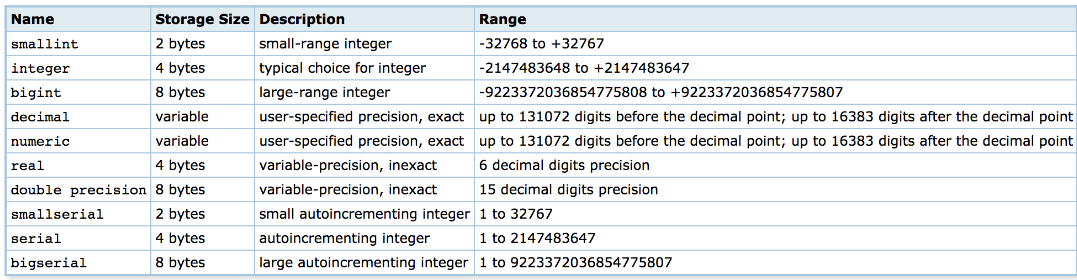
    company\_parent.ticker) = fortune500.ticker

ORDER BY rank

#### Column types and constraints



#### Types of data



* Example:

SELECT '3.2'::numeric,

       '-123'::numeric,

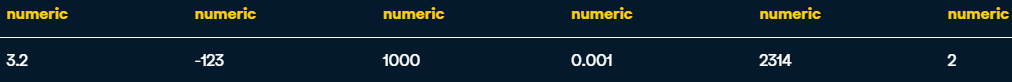
       '1e3'::numeric,

       '1e-3'::numeric,

       '02314'::numeric,

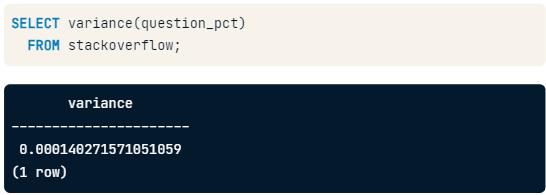
       '0002'::numeric;

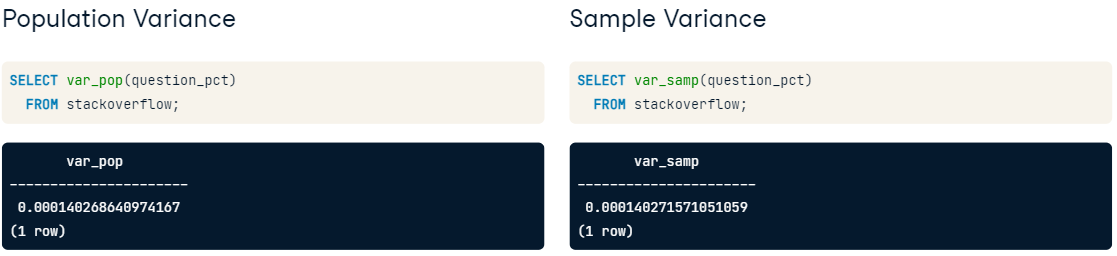
Result:



#### Numeric data type

* **Variance** == stddev\*\*2:





* Example of code:

SELECT

     sector,

     AVG(revenues/employees::numeric) AS avg\_rev\_employee

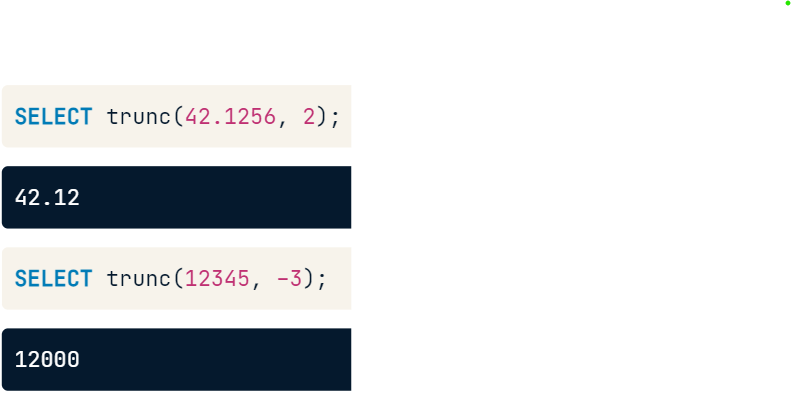
FROM fortune500

GROUP BY sector

ORDER BY avg\_rev\_employee;

#### Exploring distributions

* **Trunc**() – similar AS ROUND



* Example of the code:

SELECT trunc(employees, -4) AS employee\_bin,

  count(\*)

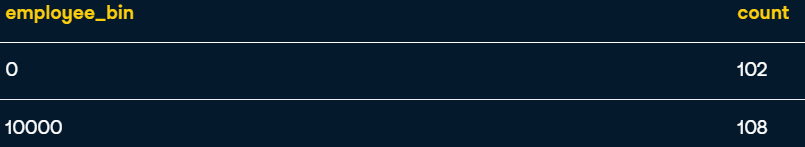
FROM fortune500

WHERE employees < 100000

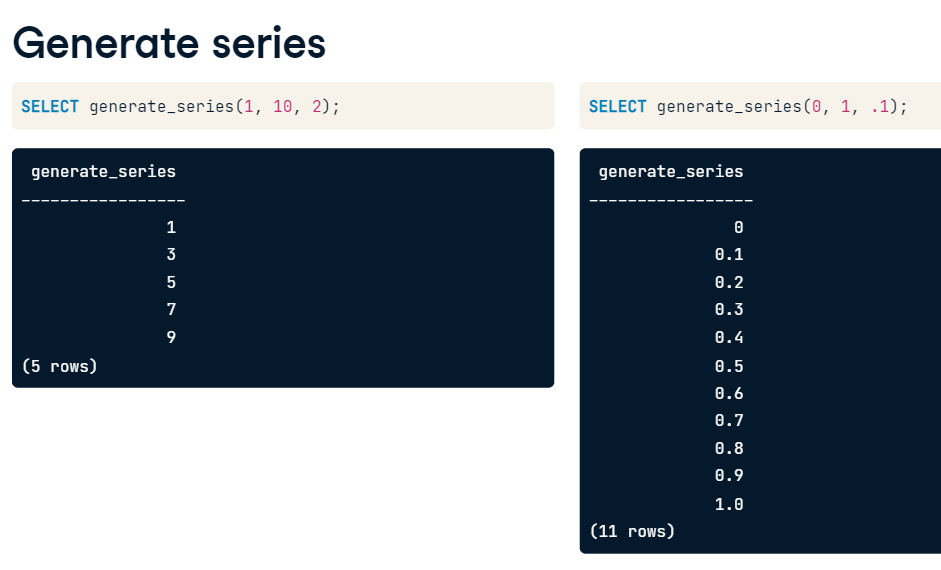
GROUP BY employee\_bin

ORDER BY employee\_bin

Result:



* Generate\_series():



* Generating bins:

WITH bins AS (

      SELECT generate\_series(2200, 3050, 50) AS lower,

             generate\_series(2250, 3100, 50) AS upper),

     dropbox AS (

      SELECT question\_count

        FROM stackoverflow

       WHERE tag='dropbox')

SELECT lower, upper, count(question\_count)

    FROM bins

    LEFT JOIN dropbox

    ON question\_count >= lower AND question\_count < upper

 GROUP BY lower, upper

 ORDER BY lower;

Result:



#### Summary functions

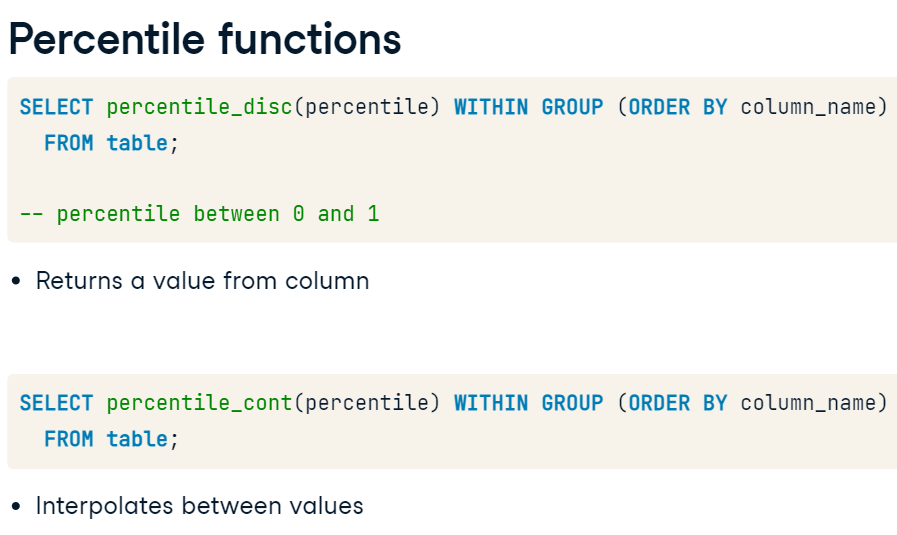
* Corr() – finds correlation between values

SELECT corr(revenues, profits) AS rev\_profits,

       corr(revenues, assets) AS rev\_assets,

       corr(revenues, equity) AS rev\_equity

  FROM fortune500;



* **Percentile\_disc** – finding discrete percenttile:

SELECT

  AVG(assets) AS mean,

  percentile\_disc(0.5) WITHIN GROUP (ORDER BY assets) AS median,

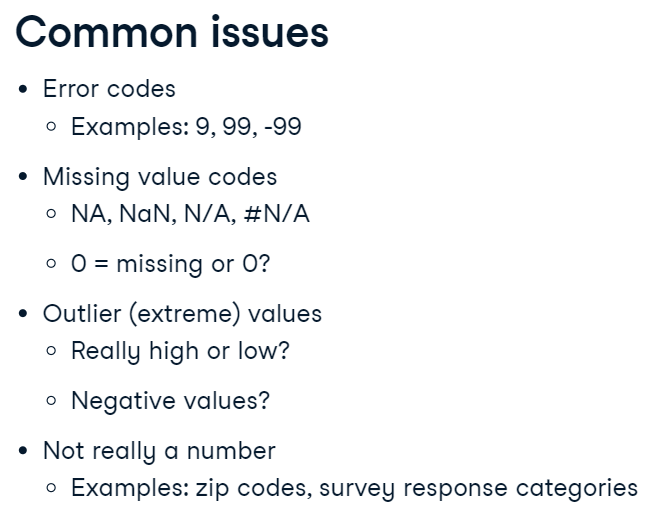
  sector

FROM fortune500

GROUP BY sector

ORDER BY mean

* Common issues:



### Creating temporary tables

* Creating and deleting table:

DROP TABLE IF EXISTS profit80;

CREATE TEMP TABLE profit80 AS

  SELECT sector,

         percentile\_disc(0.8) WITHIN GROUP (ORDER BY profits) AS pct80

    FROM fortune500

   GROUP BY sector;

SELECT

       Distinct title,

       fortune500.sector,

       profits,

       profits/pct80 AS ratio

FROM fortune500

left join profit80

on fortune500.sector = profit80.sector

where profits > pct80

result:



* Another example:

DROP TABLE IF EXISTS startdates;

CREATE TEMP TABLE startdates AS

SELECT tag, min(date) AS mindate

  FROM stackoverflow

 GROUP BY tag;

SELECT

      startdates.tag,

      mindate,

      so\_min.question\_count AS min\_date\_question\_count,

      so\_max.question\_count AS max\_date\_question\_count,

      so\_max.question\_count - so\_min.question\_count AS change

FROM startdates

INNER JOIN stackoverflow AS so\_min

  ON startdates.tag = so\_min.tag

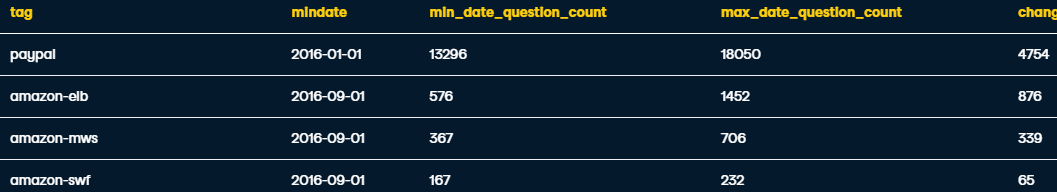
  AND startdates.mindate = so\_min.date

INNER JOIN stackoverflow AS so\_max

  ON startdates.tag = so\_max.tag

  AND so\_max.date = '2018-09-25';

Result:



* Inserting values and creating corr-matrix:

DROP TABLE IF EXISTS correlations;

CREATE TEMP TABLE correlations AS

SELECT 'profits'::varchar AS measure,

       corr(profits, profits) AS profits,

       corr(profits, profits\_change) AS profits\_change,

       corr(profits, revenues\_change) AS revenues\_change

  FROM fortune500;

INSERT INTO correlations

SELECT 'profits\_change'::varchar AS measure,

       corr(profits\_change, profits) AS profits,

       corr(profits\_change, profits\_change) AS profits\_change,

       corr(profits\_change, revenues\_change) AS revenues\_change

  FROM fortune500;

INSERT INTO correlations

SELECT 'revenues\_change'::varchar AS measure,

       corr(revenues\_change, profits) AS profits,

       corr(revenues\_change, profits\_change) AS profits\_change,

       corr(revenues\_change, revenues\_change) AS revenues\_change

  FROM fortune500;

SELECT measure,

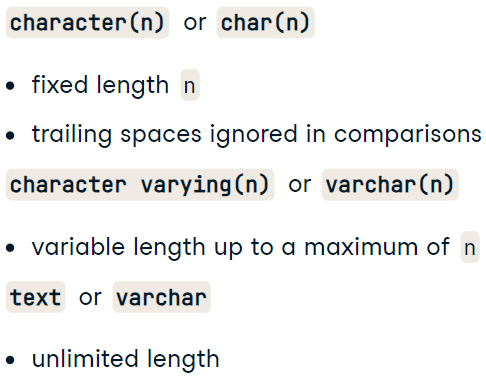
    ROUND(profits::numeric, 2) AS profits,

    ROUND(profits\_change::numeric, 2) AS profits\_change,

    ROUND(revenues\_change::numeric, 2) AS revenues\_change

FROM correlations

### Character data types





* Extra-trim – can delete any designated symbol:

SELECT DISTINCT street,

    TRIM(street, ' 0123456789#/.') AS cleaned\_street

FROM evanston311

ORDER BY street

* ILIKE – non-sensenive LIKE (whould seek for “Apple” as well as for “apple”):

SELECT category,

  COUNT(\*)

  FROM evanston311

  WHERE (description ILIKE '%trash%'

    OR description ILIKE '%garbage%')

   AND category NOT LIKE '%Trash%'

   AND category NOT LIKE '%Garbage%'

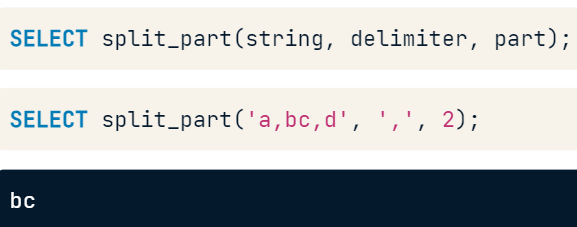
 GROUP BY category

 ORDER BY count DESC

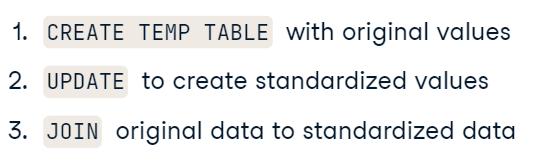
 LIMIT 10;

#### Splitting and concatenating text

* split\_part:



#### Multiple transformations



* example of code:

DROP TABLE IF EXISTS recode;

CREATE TEMP TABLE recode AS

  SELECT DISTINCT category,

         rtrim(split\_part(category, '-', 1)) AS standardized

  FROM evanston311;

UPDATE recode SET standardized='Trash Cart'

 WHERE standardized LIKE 'Trash%Cart';

UPDATE recode SET standardized='Snow Removal'

 WHERE standardized LIKE 'Snow%Removal%';

UPDATE recode SET standardized='UNUSED'

 WHERE standardized IN ('THIS REQUEST IS INACTIVE...Trash Cart',

               '(DO NOT USE) Water Bill',

               'DO NOT USE Trash', 'NO LONGER IN USE');

SELECT standardized,

  count(\*)

FROM evanston311

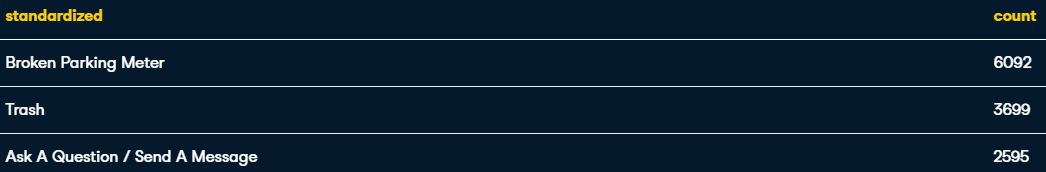
LEFT JOIN recode

ON evanston311.category = recode.category

GROUP BY standardized

ORDER BY count DESC

Result:



* seeking for values inside the SELECT:

DROP TABLE IF EXISTS indicators;

CREATE TEMP TABLE indicators AS

  SELECT id,

         CAST (description LIKE '%@%' AS integer) AS email,

         CAST (description LIKE '%\_\_\_-\_\_\_-\_\_\_\_%' AS integer) AS phone

    FROM evanston311;

SELECT priority,

  sum(email)/count(\*)::numeric AS email\_prop,

  sum(phone)/count(\*)::numeric AS phone\_prop

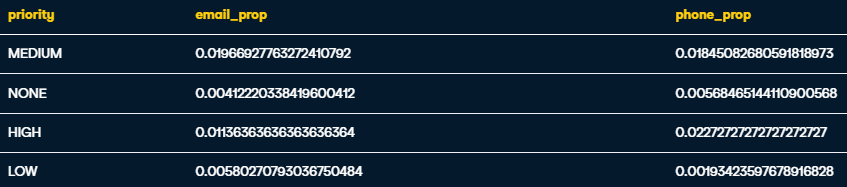
FROM evanston311

LEFT JOIN indicators

  ON evanston311.ID = indicators.id

GROUP BY priority;

Result:



### Date/time types and formats

* Casting date from datesrampt to data to easily compare values or sum:

SELECT count(\*)

  FROM evansron311

  WHERE date\_created::date = '2017-01-31'

* To\_char() – we can get the name of the day of the week by converting a timestamp to character data:

SELECT to\_char(date\_created, 'month') AS day,

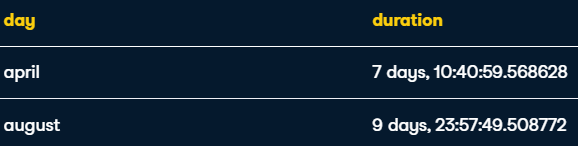
     AVG(date\_completed - date\_created) AS duration

FROM evanston311

GROUP BY day, EXTRACT(DOW FROM date\_created)

ORDER BY EXTRACT(DOW FROM date\_created)

Result:



* Trunc:

SELECT

  date\_trunc('month', day) AS month,

  AVG(count)

FROM (

  SELECT date\_trunc('day', date\_created) as day,

    count(\*) AS count

  FROM evanston311

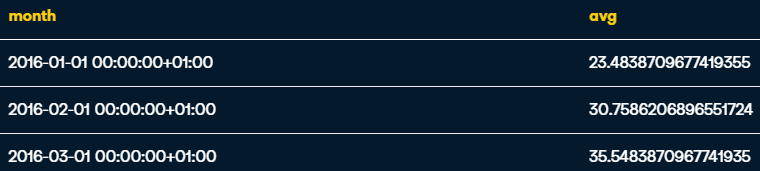
  GROUP BY day

) AS daily\_count

GROUP BY month

ORDER BY month

Result:



* Finding missing values with generate\_series:

SELECT day

FROM (SELECT

      generate\_series(MIN(date\_created), MAX(date\_created), '1 days'::interval)::date AS day

      FROM evanston311) AS all\_dates

WHERE day NOT IN (

  SELECT date\_created::date

  FROM evanston311

)

* Counting and agg values in bins:

WITH bins AS (

      SELECT generate\_series('2016-01-01',

                            '2018-01-01',

                            '6 months'::interval) AS lower,

            generate\_series('2016-07-01',

                            '2018-07-01',

                            '6 months'::interval) AS upper),

     daily\_counts AS (

     SELECT day, count(date\_created) AS count

       FROM (SELECT generate\_series('2016-01-01',

                                    '2018-06-30',

                                    '1 day'::interval)::date AS day) AS daily\_series

            LEFT JOIN evanston311

            ON day = date\_created::date

      GROUP BY day)

SELECT

     lower, upper,

     percentile\_disc(0.5) WITHIN GROUP (ORDER BY count) AS median

FROM bins

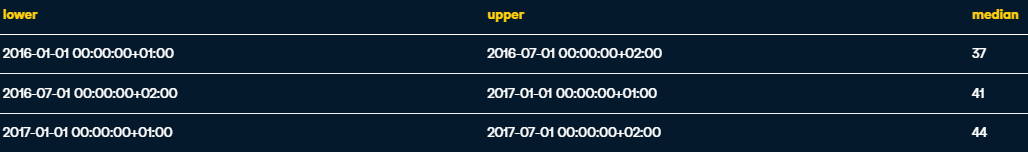
LEFT JOIN daily\_counts

     ON day >= lower and day < upper

GROUP BY lower, upper

ORDER BY lower

Result:



* Taking into consederation all values, and those, which are NULL:

WITH all\_days AS (

     SELECT generate\_series('2016-01-01',

                         '2018-06-30',

                         '1 day'::interval) AS date

),

daily\_count AS (

     SELECT date\_trunc('day', date\_created) AS day,

          count(\*)

     FROM evanston311

     GROUP BY day

)

SELECT f('month', date) AS month,

     AVG(COALESCE(count, 0)) as average

FROM all\_days

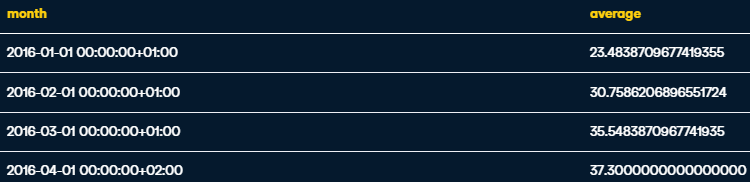
LEFT JOIN daily\_count

     ON all\_days.date = daily\_count.day

GROUP BY month

ORDER BY month

Result:



### Lag and Lead in dates

* Code:

WITH request\_gaps AS (

        SELECT date\_created,

             lag(date\_created) OVER (ORDER BY date\_created) AS previous,

             date\_created - lag(date\_created) OVER (ORDER BY date\_created) AS gap

          FROM evanston311)

SELECT \*

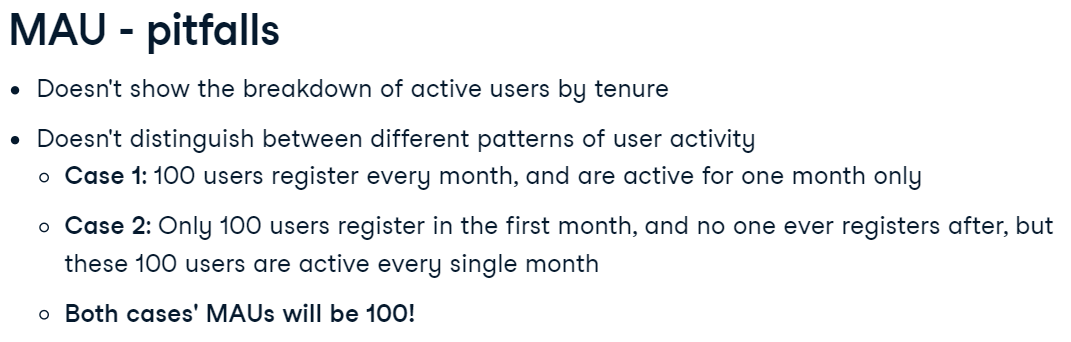
  FROM request\_gaps

 WHERE gap = (SELECT max(gap)

                FROM request\_gaps);

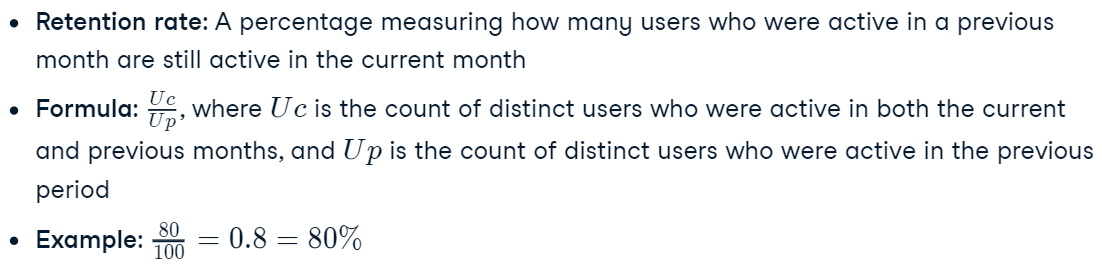
## Analyzing Business

### Patterns of behaviour



### Retention

* What is it?



* Code:

WITH user\_monthly\_activity AS (

  SELECT DISTINCT

    DATE\_TRUNC('month', order\_date) :: DATE AS delivr\_month,

    user\_id

  FROM orders)

SELECT

  previous.delivr\_month,

  ROUND(

    COUNT(DISTINCT current.user\_id)::NUMERIC

      / COALESCE(COUNT(DISTINCT previous.user\_id), 1)

        , 2) AS retention\_rate

FROM user\_monthly\_activity AS previous

LEFT JOIN user\_monthly\_activity AS current

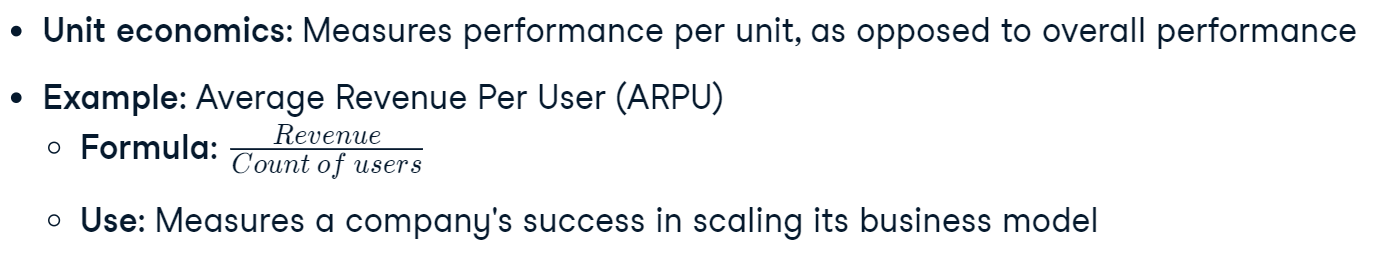
  ON previous.delivr\_month = current.delivr\_month - '1 month'::interval

    AND previous.user\_id = current.user\_id

GROUP BY previous.delivr\_month

ORDER BY previous.delivr\_month ASC

### ARPU



* How to calculate (1):

WITH KPI AS (

  SELECT

    user\_id,

    SUM(meal\_price \* order\_quantity) AS revenue

  FROM orders

  JOIN meals

    USING(meal\_id)

  GROUP BY user\_id

)

SELECT

  ROUND(AVG(revenue)::numeric,  2) AS arpu

FROM kpi

* How to calculate (2):

WITH kpi AS (

  SELECT

    DATE\_TRUNC('week', order\_date)::date AS delivr\_week,

    SUM(order\_quantity\*meal\_price) AS revenue,

    COUNT(DISTINCT user\_id) AS users

  FROM meals AS m

  JOIN orders AS o ON m.meal\_id = o.meal\_id

  GROUP BY delivr\_week)

SELECT

  delivr\_week,

  ROUND(

    revenue::numeric / GREATEST(users, 1),

    2) AS arpu

FROM kpi

ORDER BY delivr\_week ASC;

#### Histograms

* Code:

WITH user\_orders AS (

  SELECT

    user\_id,

    COUNT(DISTINCT order\_id) AS orders

  FROM orders

  GROUP BY user\_id)

SELECT

  orders,

  COUNT(user\_id) AS users

FROM user\_orders

GROUP BY orders

ORDER BY orders ASC;

#### Bucketing

* Code:

WITH user\_revenues AS (

    SELECT

      DISTINCT user\_id,

      SUM(order\_quantity\*meal\_price) AS revenue

    FROM orders AS o

    LEFT JOIN meals AS m

      USING(meal\_id)

    GROUP BY user\_id

)

SELECT

  CASE WHEN revenue < 150 THEN 'Low-revenue users'

    WHEN revenue < 300 THEN 'Mid-revenue users'

    ELSE 'High-revenue users'

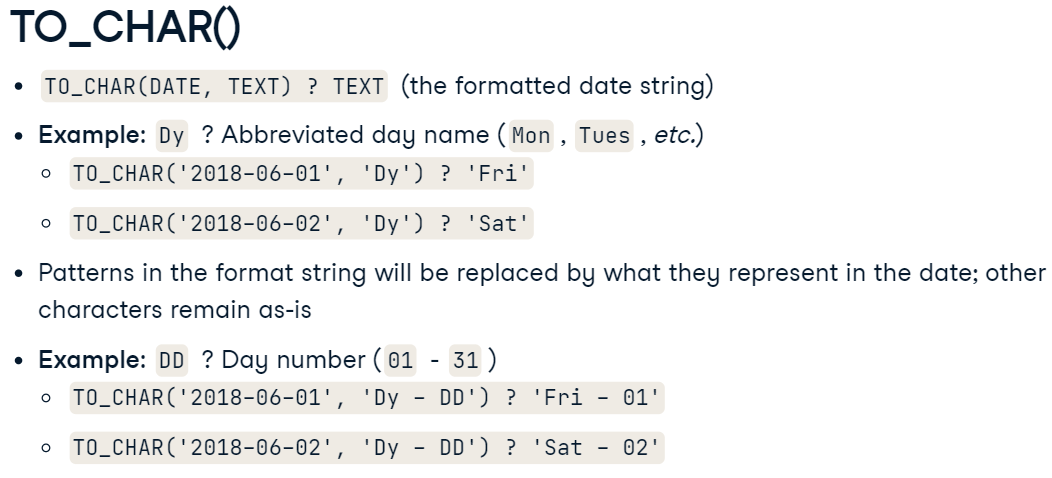
    END AS revenue\_group,

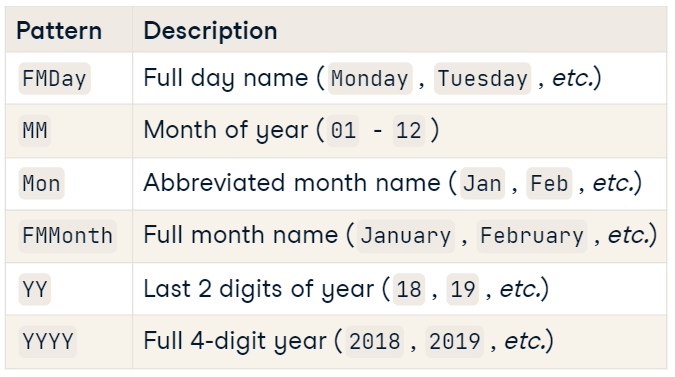
  COUNT(user\_id) AS users

FROM user\_revenues

GROUP BY revenue\_group

### Dealing with dates





* Code:

SELECT

    DISTINCT order\_date,

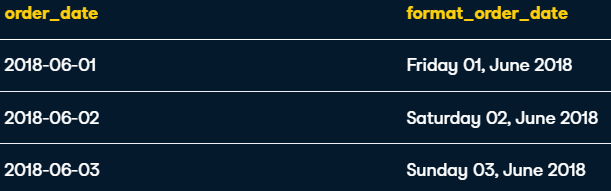
    TO\_CHAR(order\_date, 'FMDay DD, FMMonth YYYY') AS format\_order\_date

FROM orders

ORDER BY order\_date

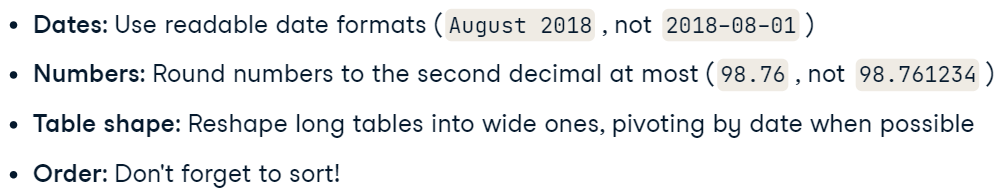
LIMIT 3

Result:



### Report readibility

* Readibility:



* CODE:

CREATE EXTENSION IF NOT EXISTS tablefunc;

SELECT \* FROM CROSSTAB($$

  WITH eatery\_users AS  (

    SELECT

      eatery,

      TO\_CHAR(DATE\_TRUNC('quarter', order\_date), '"Q"Q YYYY') AS delivr\_quarter,

      COUNT(DISTINCT user\_id) AS users

    FROM orders

    LEFT JOIN meals

      USING(meal\_id)

    GROUP BY eatery, delivr\_quarter

    ORDER BY delivr\_quarter)

    SELECT

      eatery,

      delivr\_quarter,

      RANK() OVER(PARTITION BY delivr\_quarter

                  ORDER BY users DESC)::INT AS users\_rank

    FROM eatery\_users

    ORDER BY eatery

  $$)

  AS ct (

      eatery TEXT,

      "Q2 2018" INT,

      "Q3 2018" INT,

      "Q4 2018" INT

);