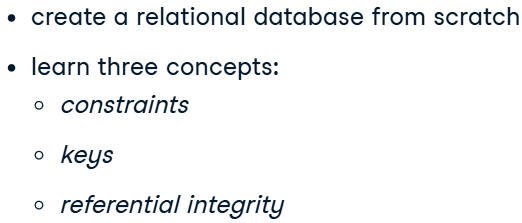
# SQL\_DB

## Introduction to Relational Databases in SQL

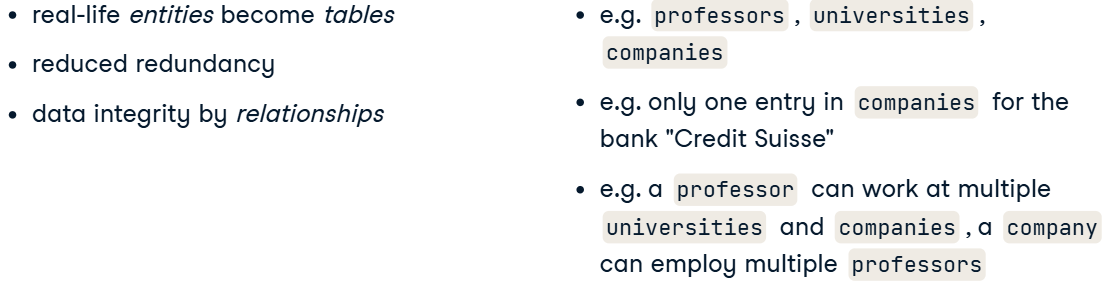
### Your first database

* Course Objection:



#### Introduction to relational databases

* Benefits of a relational database:



* Having a look at the database:

select table\_schema, table\_name

from information\_schema.tables

where table\_schema = 'public'

* Columns of a ceratin table:

select \*

from information\_schema.columns

where table\_schema = 'public' and table\_name = 'university\_professors'

* Learn table constains:

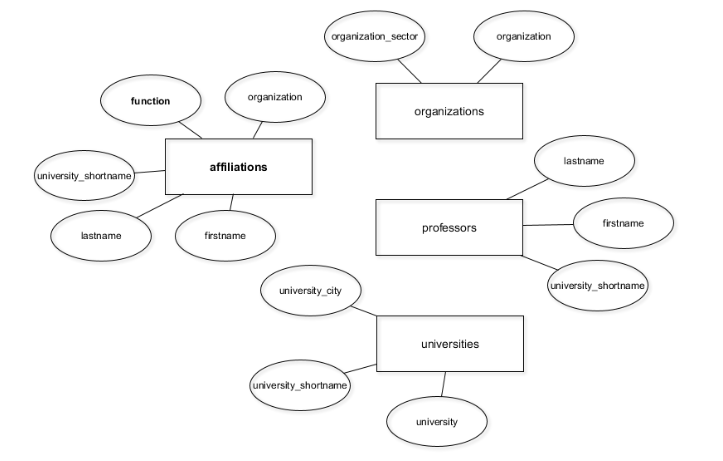
SELECT constraint\_name, table\_name, constraint\_type

FROM information\_schema.table\_constraints

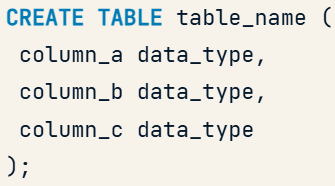
WHERE constraint\_type = 'FOREIGN KEY';

#### Tables: At the core of every database

* Entity-telationship diagram:



* + Squares denote so-called entity types;
  + Circles connected to these denote attributes (or columns).
* Creating table:



* + Code:

create table universities (

    university\_shortname text,

    university text,

    university\_city text

);

* Code for adding columns to DB:

alter table professors

add column university\_shortname text;

#### Update your database as the structure changes

* Inserting data into the new table:

insert into affiliations

select distinct firstname,

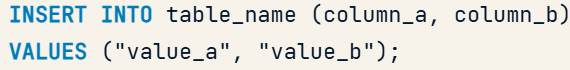
                lastname,

                function,

                organization

from university\_professors;

* Inserting into table manually:



* Renaming column in a table:

alter table affiliations

rename column organisation to organization;

* Dropping columns in a table:

alter table affiliations

drop column university\_shortname;

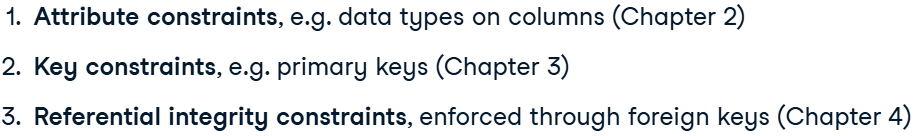
* Dropping table:

drop table university\_professors;

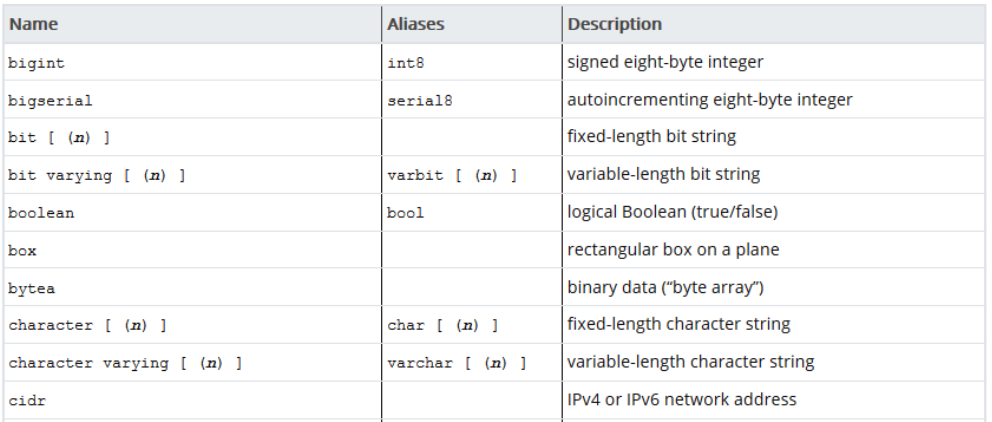
### Enforce data consistency with attribute constraints

#### Better data quality with constraints

* Integrity constrains (Обмеження цілісності):
  + Types:

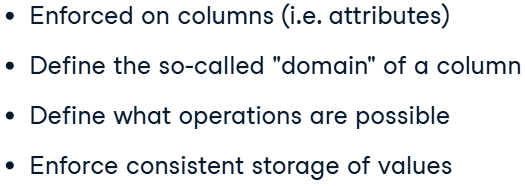


* + - **attribute constraints**, which enforce data types for columns;
    - **key constraints**, such as primary keys that ensure unique records;
    - **referential integrity constraints**, which link tables together.
* Data types as *attribute constrains*:

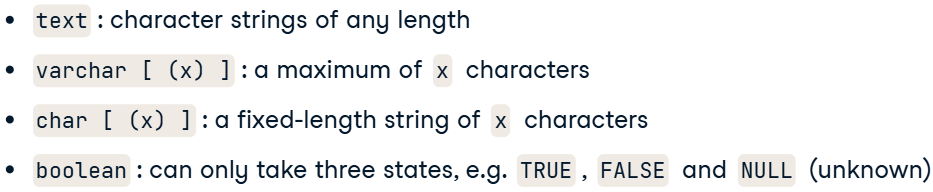


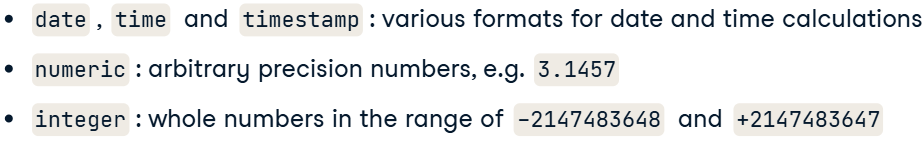
#### Working with data types

* Working with data types:



* + "domain" of values in a column, that means, what form these values can take – and what not.
* Common types:





* Altering data type after table creation:

alter table professors

alter column firstname

type varchar(64)

* + Then you can use the "USING" keyword, and specify a transformation that should happen before the type is altered:

alter table professors

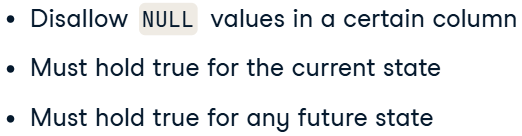
alter column firstname

type varchar(16)

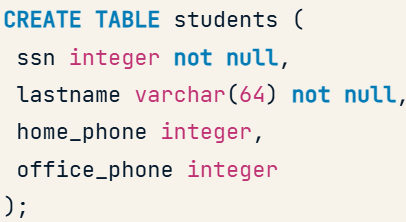
using substring(firstname from 1 for 16)

#### The not-null and unique constraints

* What is a **not-null** **constaint**:



* + you can specify a not-null constraint on a columns that doesn’t have Nulls.
* Adding not-null constraint:



* Adding not-null constraint after the table is created:
  + add not null:

alter table professors

alter column lastname

set not null

* + drop not null:

alter table professors

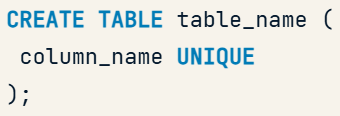
alter column lastname

drop not null

* **Unique constraint**:



* Creating not­–null constraint:



* Adding unique constraint after the table is created:

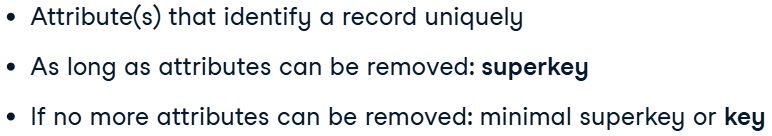
alter table universities

add constraint university\_shortname\_unq unique(university\_shortname)

### Uniquely identify records with key constraints

#### Keys and superkeys

* What is a key:



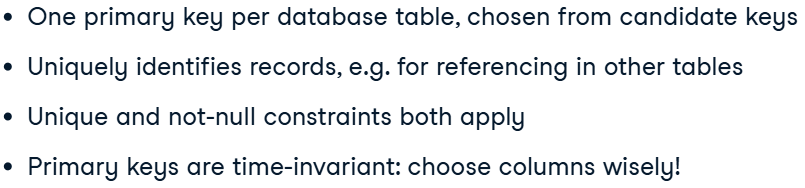
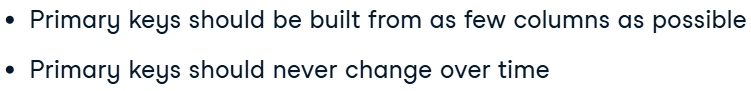
* + superkey – это **набор столбцов**, который **уникально идентифицирует** каждую строку в таблице.
* How to find primary key:
  + Count the distinct records for all possible combinations of columns. If the resulting number x equals the number of all rows in the table for a combination, you have discovered a *superkey*.
  + Then remove one column after another until you can no longer remove columns without seeing the number x decrease. If that is the case, you have discovered a *(candidate) key*.
  + Code:

select count(distinct(firstname, lastname))

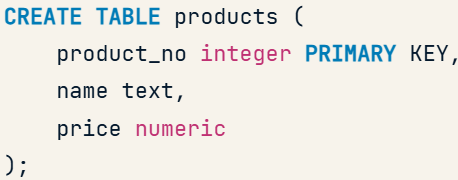
from professors

#### Primary keys

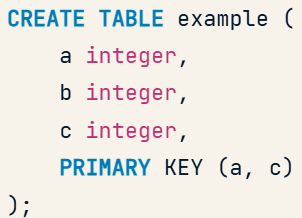
* Primary key:
  + What is it:

* + Code:



* + Code for more than one column in a key:



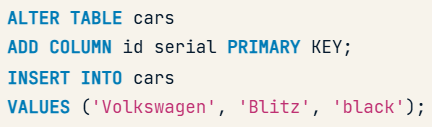
* Code for specifying primary keys in created table:

alter table organizations

add constraint organization\_pk primary key (id)

#### Surrogate keys

* Surrogate key – artificial primary key, which is not based on a native columns of the DB:
* Code:



* + Alternative:

select count(distinct(make, model))

from cars;

alter table cars

add column id varchar(128);

update cars

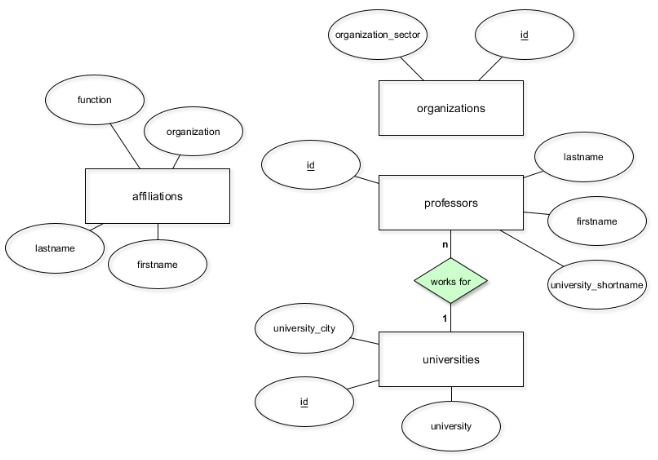
set id = concat(make, model);

* + - **ALTER TABLE** – изменяет структуру таблицы (добавляет, удаляет, изменяет столбцы).
    - **UPDATE** – изменяет данные в таблице.

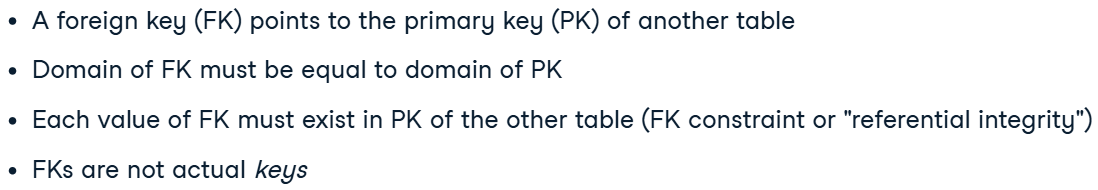
### Glue together tables with foreign keys

#### Model 1:N relationships with foreign keys

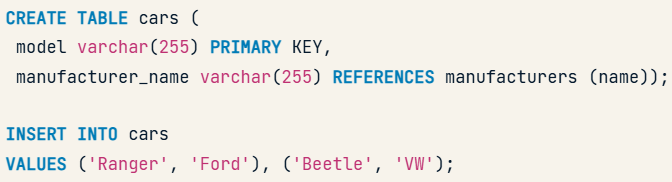
* ER model:



* Foreign key – a key that points to primary key of another table:
  + Внешний ключ в SQL гарантирует, что ссылка на другую таблицу допустима: значение должно существовать в первичном ключе связанной таблицы. Если такого значения нет, вставка или обновление приведут к ошибке.
  + Details:



* Creating foreign key:



* Foreign key for existing table:

alter table professors

add constraint professors\_fkey foreign key (university\_id)

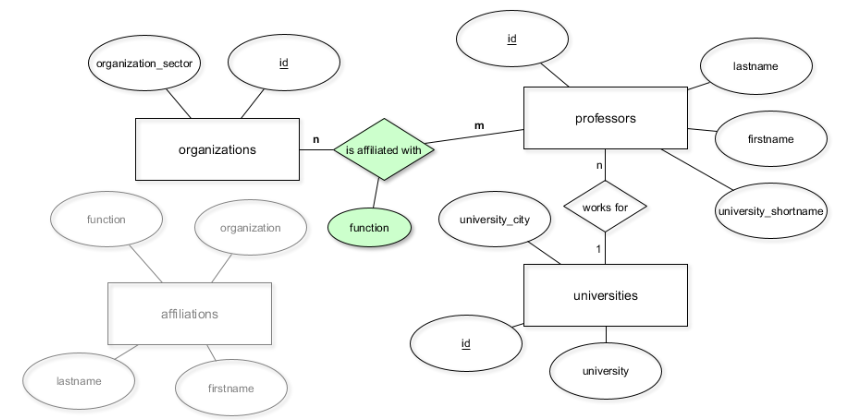
    references universities (id)

* Naming convention:

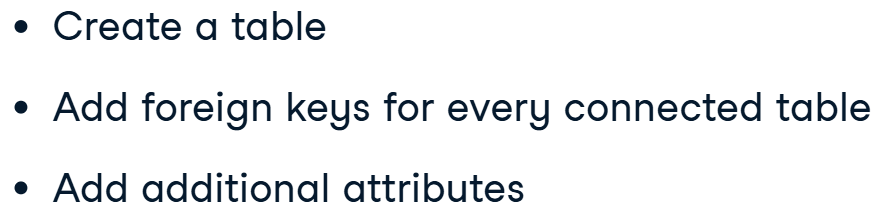
Pay attention to the **naming convention** employed here: Usually, a foreign key referencing another primary key with name id is named x\_id, where x is the name of the referencing table in the singular form.

#### Model more complex relationships

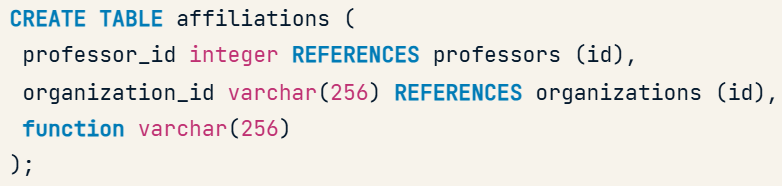
* ER model:



* + Old table affiliations no longer needed.
* Implementing N:M relationship:



* + - No primary key.
* Code:



* Code for filling values in the table, which is referencing to another:

alter table affiliations

add column professor\_id integer references professors(id);

update affiliations

set professor\_id = professors.id

from professors

where affiliations.firstname = professors.firstname

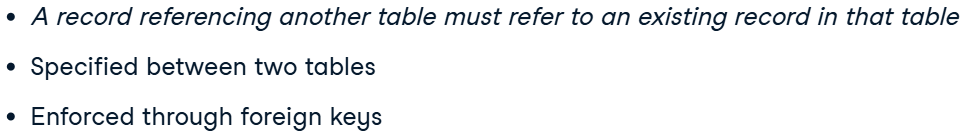
    and affiliations.lastname = professors.lastname

* + Explonation:

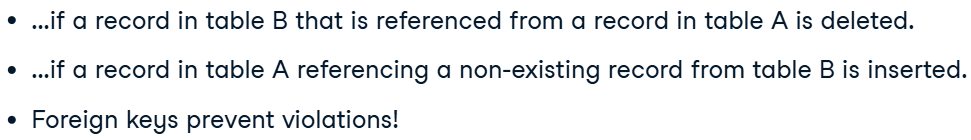
The firstname and lastname columns of affiliations were used to establish a link to the professors table in the last exercise – so the appropriate professor IDs could be copied over.

#### Referential integrity

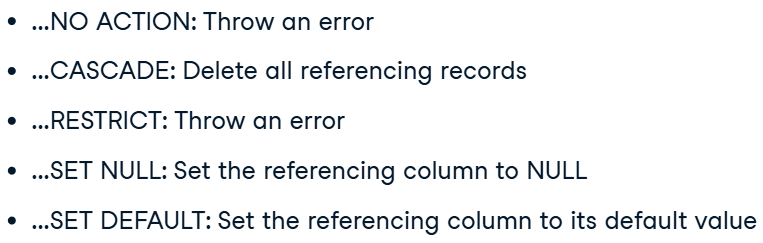
* What is a referentail integrity:



* When violating:



* Dealing with violations (1 – restict deletion; 2 – deletion will occur in the second table too):



* Example (dropping constaint and chanching with a new one):

alter table affiliations

drop constraint affiliations\_organization\_id\_fkey;

alter table affiliations

add constraint affiliations\_organization\_id\_fkey

    foreign key (organization\_id) references organizations(id)

    on delete CASCADE;

## Introduction to BigQuery

### BigQuery Architecture and Structure

#### BigQuery data organization

* How the data is organized:



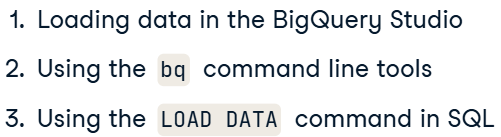
* + from project;
  + from dataset, which contains many tables;
  + from table.
* Addintional info:

Tables in BigQuery follow a hierarchy that allows them to have permissions and access rights applied to them at three different levels. This pertains not only to the access rights assigned within Google Cloud but also to the physical location of your data and the resources you will have access to.

### Writing queries and data types

#### Data ingestion in BigQuery

* Loading data:



* + BigQuery Studio:



* + Load Data (only from Google Cloud Storage):

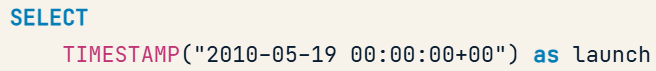


#### Date/time types in BigQuery

* Date (a single day):



* Timestamp (absolute point in time):



* Extracting Dates:

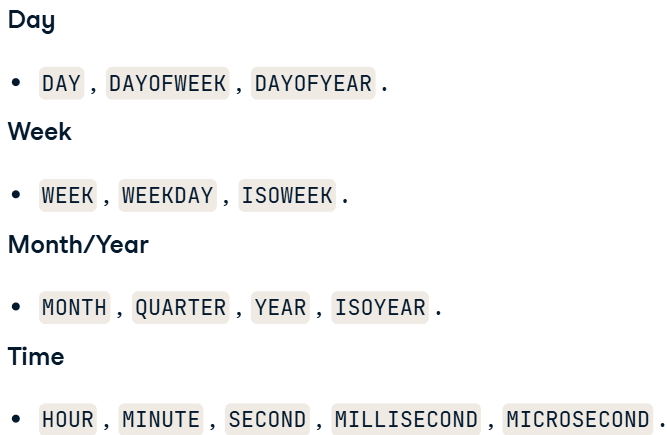
select count(order\_id)

from ecommerce.ecomm\_order\_details

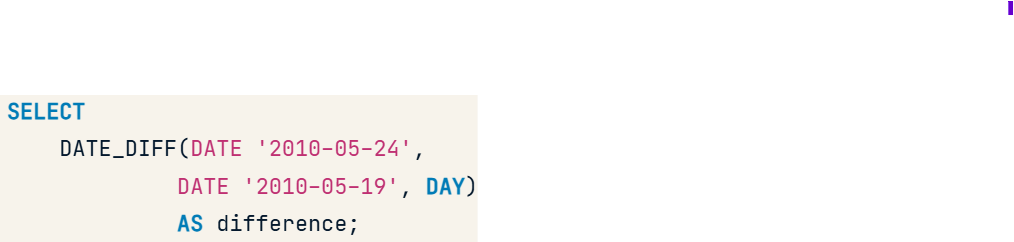
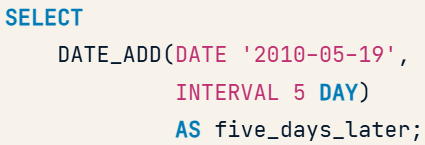
where extract(QUARTER from order\_purchase\_timestamp) = 4

    and extract(YEAR from order\_purchase\_timestamp) = 2017

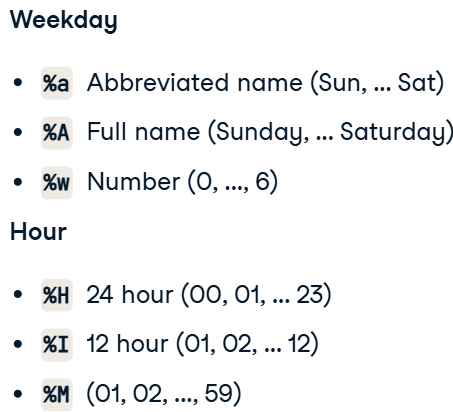
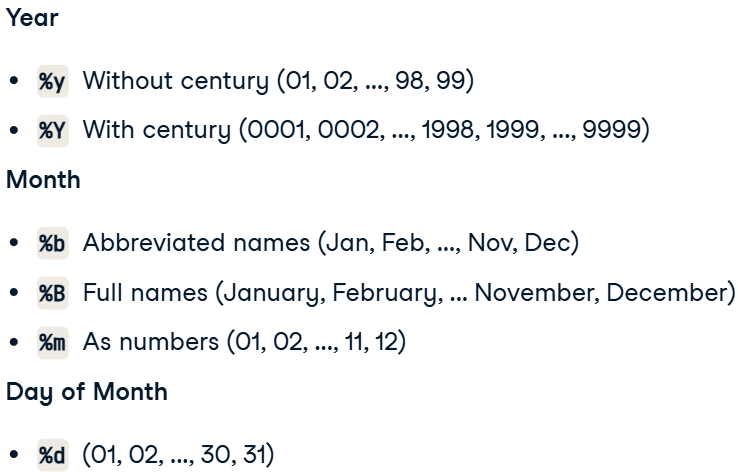
* Date and Timestamp parts (with EXTRACT):

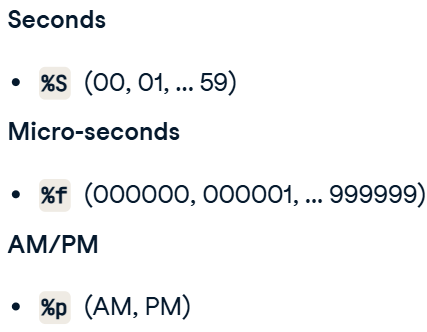
 + date

* Functions on datetimes:
  + date(timestamp)\_add, date(timestamp)\_subtract, date(timestamp)\_diff.

* String format of the date:





* + Code:

select

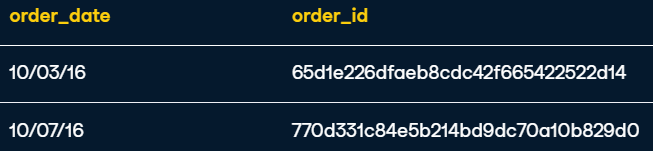
    format\_timestamp('%m/%d/%Y', order\_purchase\_timestamp) as order\_date,

    order\_id

from ecommerce.ecomm\_order\_details

limit 3

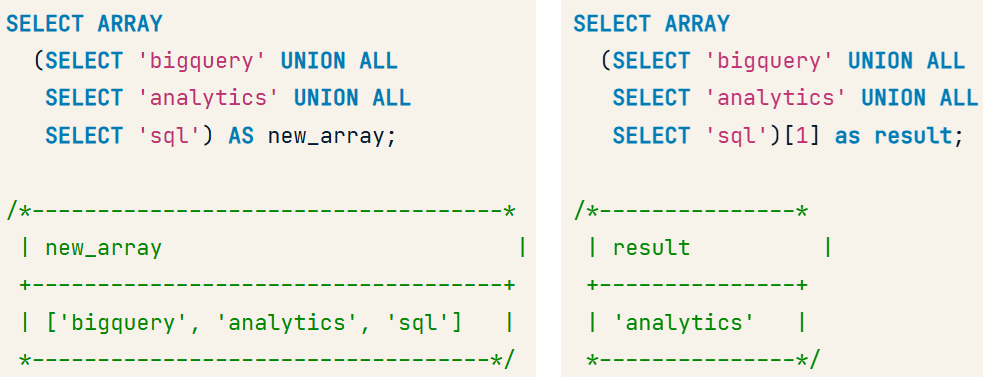
result:



* Current date:
  + current\_date();
  + current\_timestamp().

#### Unstructured data

* Arrays (like lists in python):



* + Code:

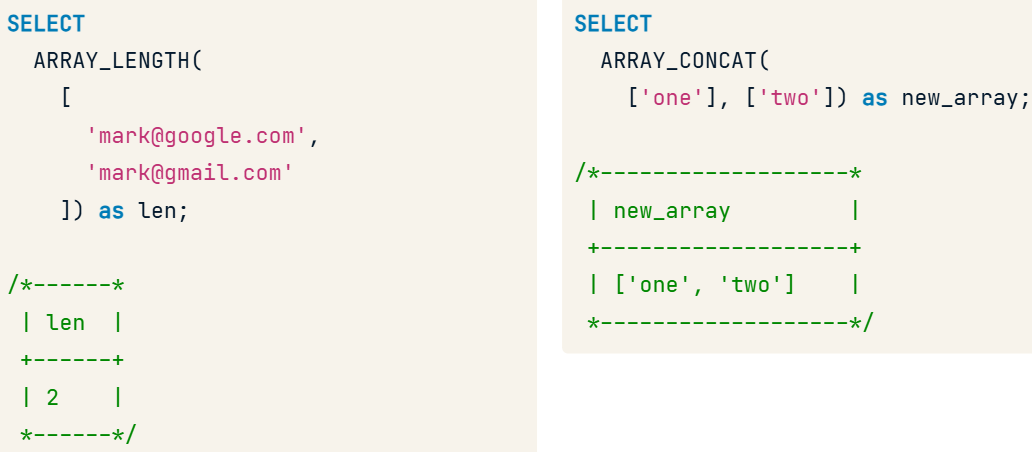
select array

    (select product\_id

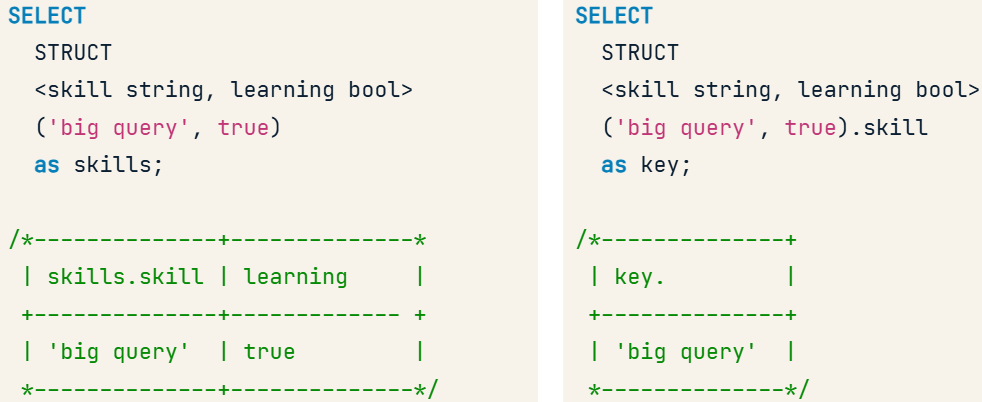
    from ecommerce.ecomm\_products

    where product\_weight\_g = 2220)

* + Additional functions:



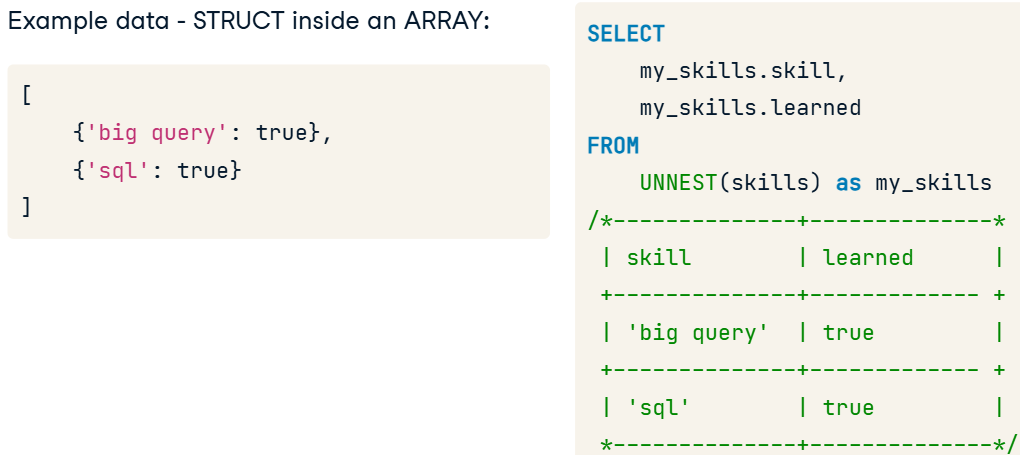
* SCTRUCTs (dictionary):
  + STRUCT can contain a single value, key-value pairs, or even nested data with ARRAYs or STRUCTs inside the STRUCT:



* Unnest – allows us to flatten ARRAYs/STRUCT and turn them into rows within a table:
  + With Arrays:



* + With Sturcts:



* + Code:

SELECT

    order\_id, items.price

FROM

    ecommerce.ecomm\_orders,

    unnest(order\_items) as items

WHERE order\_id = 'a0e747c954a595b0e3458c87ab1a4958';

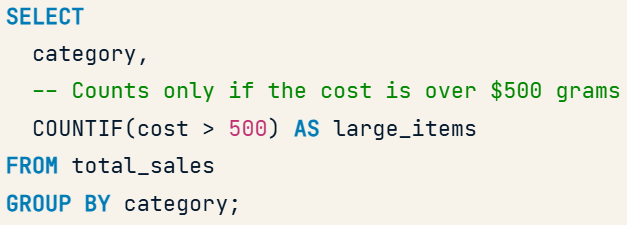
* Search(“data structrure to search in”, ‘strint\_to\_search’) – a flexible function to search across ***any*** data type, including unstructured data:
  + Code:



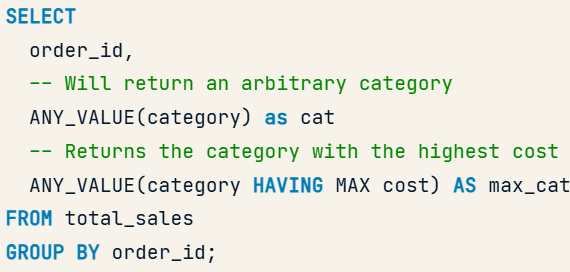
### Querying data in BigQuery

#### Aggregations

* CountIf:

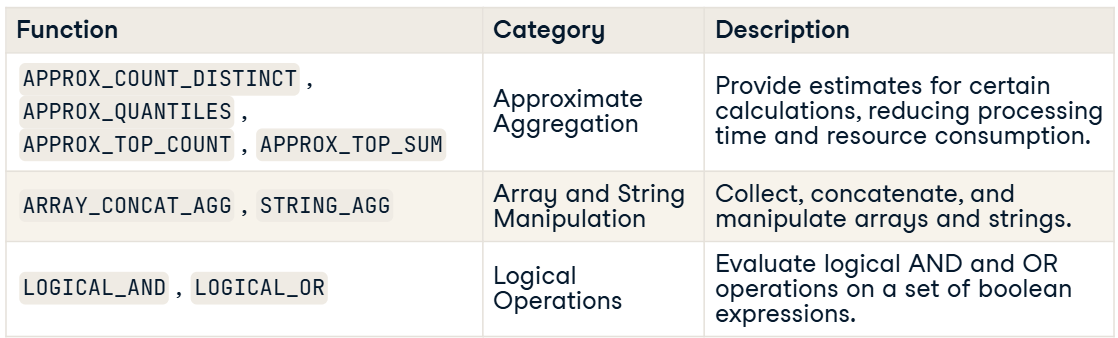


* AnyValue:



#### Special aggregations in BigQuery

* Functions (offer advantages such as reduced processing time and resource consumption, making them ideal for analyzing massive datasets):



* **Array\_Concat\_Agg(column)** – объединяет несколько массивов по колонке в один массив без вложенных внутри:

SELECT

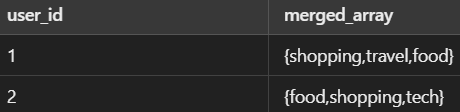
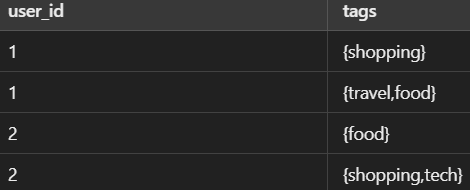
    user\_id,

    ARRAY\_CONCAT\_AGG(tags) AS merged\_array

FROM transactions

GROUP BY user\_id;

result (from – to):



* **String\_Agg(column, delimiter)** – обьединяет неколько строк (со строк) в одну.

SELECT

    o.order\_id,

-- Use the STRING\_AGG to find distinct values and separate them by a comma with a space

    STRING\_AGG(distinct product\_category\_name\_english, ', ')

FROM ecommerce.ecomm\_orders o, unnest(order\_items) items

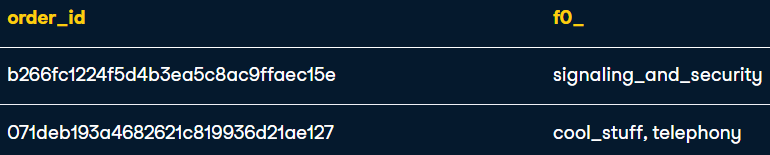
JOIN ecommerce.ecomm\_products p

    ON items.product\_id = p.product\_id

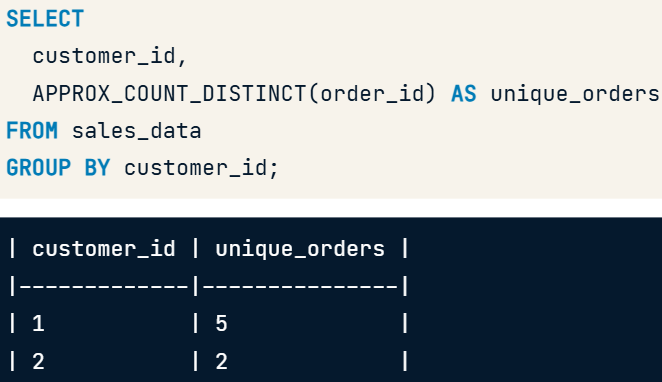
-- Find the number of items in the order\_items column

WHERE ARRAY\_LENGTH(o.order\_items) > 1

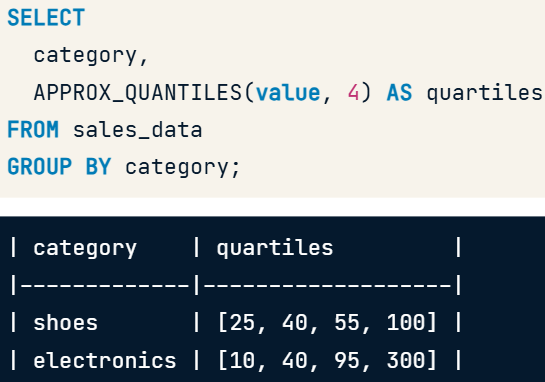
GROUP BY order\_id



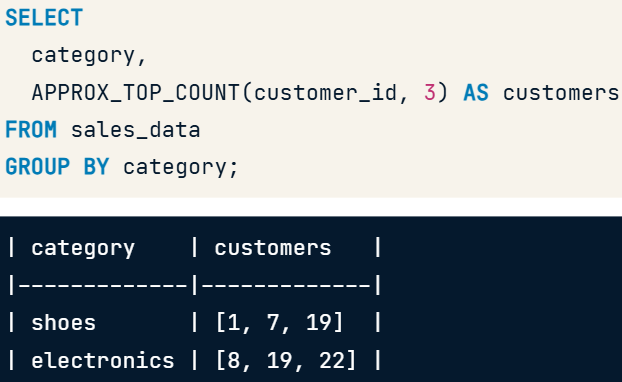
* **Approx\_Count\_Distinct(column)** – gives a quick estimation of the count of distinct values in a large DF.



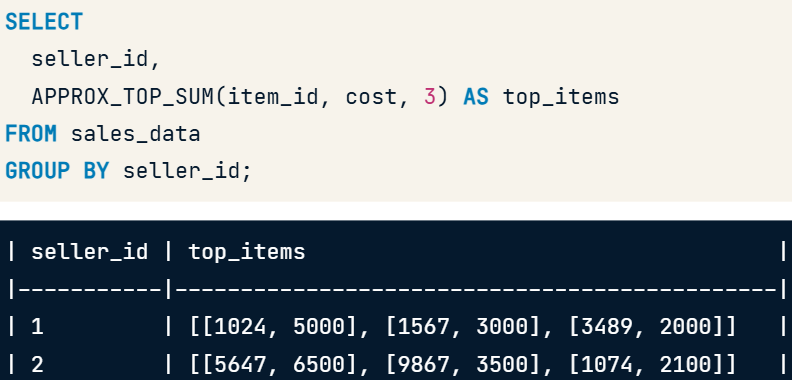
* **Approx\_Quantiles(column, bins)** – gives approximate estimate of quantiles of a numerical column in a big DF (quantiles, decentiles).



* **Approx\_Top\_Count(count, K)** – identifies the top K elements based on their occurance.



* **Approx\_Top\_Sum(el, weight, K)** – identifies the top K elements (el) based on weight.
  + Ответ в формате [item\_id, cost]



* **Logical\_AND, Logical\_OR**:
  + we use LOGICAL\_AND to see if all orders for that customer have been shipped and LOGICAL\_OR to see if at least one has shipped.

SELECT

  customer\_id,

  -- Return True, if all orders for that customer have been shiped

  LOGICAL\_AND(order\_status = 'delivered') as all\_delivered,

  -- Return True, if at least one order for that customer have been shiped

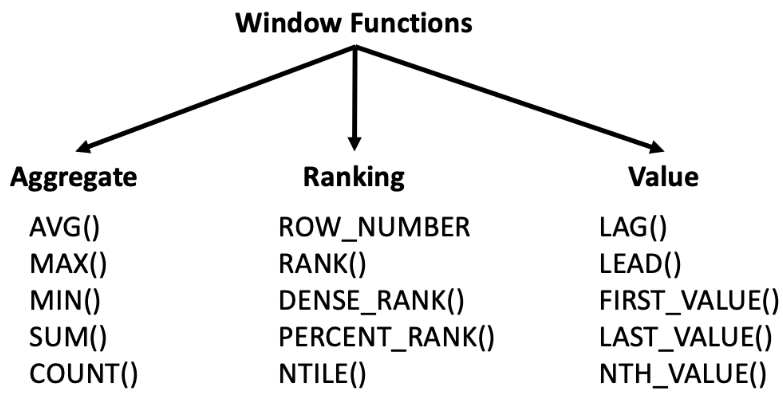
  logical\_or(order\_status = ‘shipped’) as any\_shipped

FROM ecommerce.ecomm\_order\_details

GROUP BY customer\_id;

#### WINDOW Functions

* Functions:



* Percent\_Rank() – это **аналог квантильного ранжирования**, который показывает **относительное положение строки** среди всех строк в выборке:
* **Qualify** – acts as a filter in the WHERE clause to filter values based on a WINDOW condition:

SELECT

  order\_id,

  order\_purchase\_timestamp,

  -- Complete the rolling average for the current and nine previous rows

  AVG(item.price)

  OVER(ORDER BY order\_purchase\_timestamp

       ROWS BETWEEN 9 PRECEDING AND CURRENT ROW) as rolling\_avg

FROM ecommerce.ecomm\_order\_details od

JOIN ecommerce.ecomm\_orders o

USING (order\_id), unnest(o.order\_items) as item

-- Add the qualify statement to find all rows with an average over $500

QUALIFY rolling\_avg > 500

ORDER BY order\_purchase\_timestamp;

result:

