**SQL for Business Analyst (Skill Track)**

**Exploratory Data Analysis in SQL**

**Chapter1**

* Explore SQL database
* Summarize data
* Clean a messy data

**Database client** = The program to connect to, work with, a database

A diagram of a database

Description automatically generated

* **A screenshot of a computer

  Description automatically generatedEntity-relationship** diagram (**schema)**:
* Company\_id : id means, foreign keys from one table to another table
* Company\_id : id means, foreign keys from one table to another table
* Parent\_id : id means, foreign keys reference a column in the same table
* If there is not any foreign key, ir doesn’t prevent us from joining them

A screenshot of a computer

Description automatically generated

* Note that COUNT() only counts the non-null values

**The keys to the database**

Understanding formal relationships, or links between tables (Foreign keys)

A screenshot of a computer code

Description automatically generatedA white text with black text

Description automatically generated

Example:

**Table: Employees**

| **FirstName** | **MiddleName** | **LastName** |
| --- | --- | --- |
| John | NULL | Doe |
| Jane | Alice | Smith |
| Robert | NULL | Brown |
| Emily | Marie | Johnson |

**SQL Query Using COALESCE**

SELECT

FirstName,

COALESCE(MiddleName, LastName) AS PreferredName

FROM

Employees;

**Result**

| **FirstName** | **PreferredName** |
| --- | --- |
| John | Doe |
| Jane | Alice |
| Robert | Brown |
| Emily | Marie |

In this result:

* For John, since MiddleName is NULL, COALESCE uses LastName (Doe).
* For Jane, MiddleName (Alice) is used since it’s not NULL.
* For Robert, MiddleName is NULL, so LastName (Brown) is used.
* For Emily, MiddleName (Marie) is used directly.
* **Foreign keys** reference another row in the database via a unique ID. Values in a foreign key column are restricted to values in the referenced column OR NULL.

A **foreign key** in a database is a column (or set of columns) in one table that references the **primary key** of another table (or even the same table). It enforces a link between the data in the two tables, ensuring that values in the foreign key column must either:

1. Match an existing value in the referenced (primary key) column of another table.
2. Be NULL if the relationship allows it (meaning there's no reference).

**Simple Example**

Imagine you have two tables in a database: **Customers** and **Orders**.

**Customers Table**

| **customer\_id (Primary Key)** | **customer\_name** |
| --- | --- |
| 1 | Alice |
| 2 | Bob |
| 3 | Charlie |

* customer\_id is the **primary key** of the Customers table. Each customer has a unique ID.

**Orders Table**

| **order\_id** | **customer\_id (Foreign Key)** | **order\_amount** |
| --- | --- | --- |
| 101 | 1 | 200 |
| 102 | 2 | 150 |
| 103 | 4 | 300 |
| 104 | NULL | 250 |

* customer\_id in the Orders table is a **foreign key** that references customer\_id in the Customers table.
* This means that values in the Orders.customer\_id column should only contain values found in Customers.customer\_id, or be NULL.

**Explanation**

1. Row 1 in Orders has customer\_id = 1, which is valid because 1 exists in Customers.
2. Row 2 has customer\_id = 2, also valid.
3. Row 3 has customer\_id = 4, which is **invalid** because there’s no customer\_id = 4 in Customers.
4. Row 4 has customer\_id = NULL, which is **allowed** if the foreign key constraint permits NULL values.

In summary, the foreign key constraint helps ensure that each customer\_id in Orders either points to an existing customer in Customers or is NULL.

**Column Types and Constraints**

A screenshot of a computer

Description automatically generated

A close-up of a list of data

Description automatically generatedA list of words on a white background

Description automatically generated

A screenshot of a computer

Description automatically generated

* Reminder: for converting the type use CAST(), use can either use it for the value or the column

A close-up of a white background

Description automatically generated

A screenshot of a computer

Description automatically generated

A close-up of a white background

Description automatically generated

**OR:**

A screenshot of a computer program

Description automatically generated

Examples:

-- Select the original value

SELECT profits\_change,

        -- Cast profits\_change

       CAST( profits\_change AS integer) AS profits\_change\_int

  FROM fortune500;

-- Divide 10 by 3

SELECT 10/3,

       -- Cast 10 as numeric and divide by 3

       10::numeric/3;

SELECT '3.2'::numeric,

       '-123'::numeric,

       '1e3'::numeric,

       '1e-3'::numeric,

       '02314'::numeric,

       '0002'::numeric;

**Chapter2**

**Summarizing and Aggregating Numeric Data**

**Numeric Data Types and Summary Functions**

* Columns and variables (whole number and decimals)
* Serials are integer columns that autoincrement (For ID columns)

A screenshot of a calculator

Description automatically generated

A white and black text on a white background

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated



A close-up of a text

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Exploring distributions**

* Crucial for finding errors, outliers and other anomalies in data

A screenshot of a computer

Description automatically generated

* **How to make a bigger groups include more data?**
  + Truncate function: Reduce the precision of a number, replacing the smallest numeric places, the right-most digits- with zeros.

A screenshot of a computer

Description automatically generated

* + Different from Round, it only gets rid of the extra characters.
  + The second argument, positive amounts, truncates after decimals, and negative one truncates before decimals from real values.
* **Solution:**

A screenshot of a computer code

Description automatically generated

**PAY ATTENTION!**

The SQL query execution order is indeed as follows:

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

However, **aliases defined in the SELECT clause can still be used in GROUP BY and ORDER BY**. This is because SQL engines have an optimization phase where they recognize column aliases from SELECT and allow them to be referenced in GROUP BY and ORDER BY, even though SELECT logically comes later.

In other words, SQL engines "rewrite" or parse the query so that these references to aliases in GROUP BY and ORDER BY are resolved correctly, even if they appear out of logical order. This is an exception to the strict execution order to make SQL queries easier to write and read.

* How to make a smaller and desired size of the group? Making Series

A close-up of a logo

Description automatically generated

A screenshot of a computer program

Description automatically generated

* Now making bins for the dataset:

A screenshot of a computer

Description automatically generated

* The solution is to making sub-quiries as below:

A screenshot of a computer code

Description automatically generated

**More Summary Functions**

* Finding relationships between columns (correlation -1< <1) on top of the summary like counting

A graph of a graph of variable and variable

Description automatically generated

A computer code with numbers and lines

Description automatically generated

A blue background with white text

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

A white background with blue text

Description automatically generated

Example codes:

-- What groups are you computing statistics by?

SELECT sector,

       -- Select the mean of assets with the avg function

       avg(assets) AS mean,

       -- Select the median

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

  FROM fortune500

 -- Computing statistics for each what?

 GROUP BY sector

 -- Order results by a value of interest

 ORDER BY mean;

**Creating Temporary Tables**

A screenshot of a computer

Description automatically generated

* First one is recommended and allows to use option not available with the “select into”

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

* Notice in different editors, it would be shown how many columns have been added.
* Titles should be match with the one in the table to be added

A close-up of a logo

Description automatically generated

* The table will be deleted without any warning
* Temporary will be deleted when you disconnect from the database

A blue text on a white background

Description automatically generated

* It doesn’t stop the query if it doesn’t exist

Example code:

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 each column, rounding the correlations

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;

**Chapter3**

**Character data types and common issues**

* Exploring character and text

A screenshot of a computer

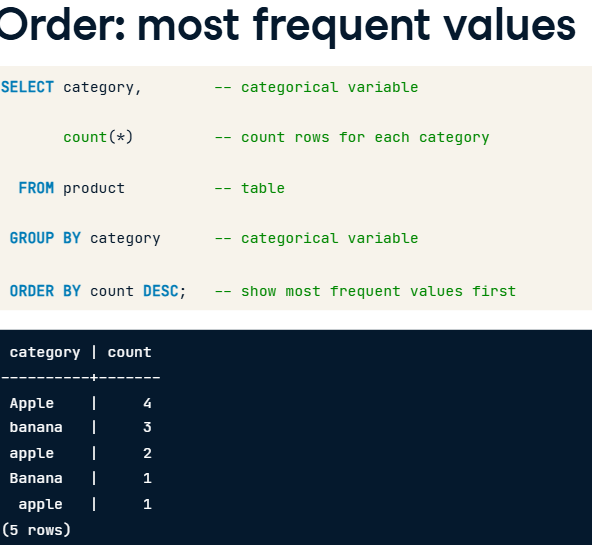
Description automatically generated

A close up of a letter

Description automatically generated

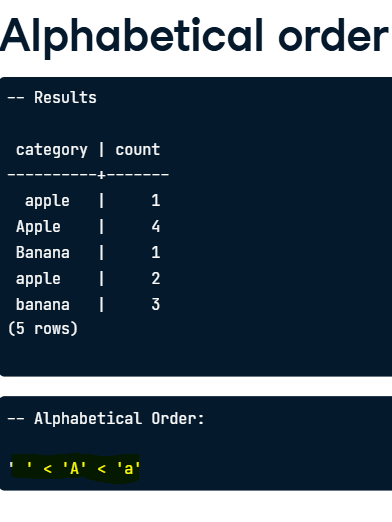
A screenshot of a computer program

Description automatically generated



A screenshot of a computer program

Description automatically generated



A screenshot of a computer

Description automatically generated

A white square with a black and blue stripe

Description automatically generated

A screenshot of a phone

Description automatically generated

* Reminder:
  + **WHERE**: Filters rows before grouping. It cannot use aggregate functions.
  + **HAVING**: Filters groups after aggregation. It can use aggregate functions.
    - * + SELECT customer\_id, SUM(amount) AS total\_amount

FROM Orders

GROUP BY customer\_id

HAVING SUM(amount) > 500;

**Cases and Spaces**

* Differences in spaces in a string and characters 🡪 Functions for changing the case and remove spaces (LIKE)
* Converting Case: lower() and upper()/ no affect on punctuation and numbers

A screenshot of a computer

Description automatically generated

* Use ILIKE, when we want to make sure upper cases are considered:

A screenshot of a computer screen

Description automatically generated

* How to clear space before and after 🡪 trim()

A screenshot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A close-up of a computer screen

Description automatically generated

Code example:

SELECT distinct street,

       -- Trim off unwanted characters from street

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

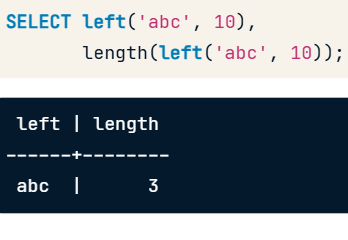
  FROM evanston311

 ORDER BY street;

**Splitting and concatenating text**

* Text values 🡪 break string into multiple pieces 🡪string new variable 🡪 join or concatenate string together





A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer code

Description automatically generated

A close-up of a logo

Description automatically generated

A screenshot of a computer

Description automatically generated

Code example:

-- Select the first 50 chars when length is greater than 50

SELECT CASE WHEN length(description) > 50

            THEN LEFT(description, 50) || '...'

       -- otherwise just select description

       ELSE description

       END

  FROM evanston311

 -- limit to descriptions that start with the word I

 WHERE description LIKE 'I %'

 ORDER BY description;

Review CASE:

In SQL, CASE is a conditional expression that allows you to apply if-then-else logic within your queries. It evaluates conditions and returns specific values based on whether each condition is met.

In the provided SQL script:

SELECT CASE

WHEN length(description) > 50

THEN LEFT(description, 50) || '...'

ELSE description

END

**Strategies for   
Multiple Transformations**

A screenshot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A close up of text

Description automatically generated

A screen shot of a computer code

Description automatically generated

A screen shot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Code Example:

-- Code from previous step **(Step0)**

DROP TABLE IF EXISTS recode;

-- **(Step1)**

CREATE TEMP TABLE recode AS

  SELECT DISTINCT category,

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

  FROM evanston311;

--(**Step2)**

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 the recoded categories and the count of each (**Step3)**

SELECT standardized, COUNT(\*)

-- From the original table and table with recoded values

  FROM evanston311

       LEFT JOIN recode

       -- What column do they have in common?

       ON evanston311.category=recode.category

 -- What do you need to group by to count?

 GROUP BY standardized

 -- Display the most common val values first

 ORDER BY count DESC;

**Chapter4**

**Date/time types and formats**

* Date and timestamp

A screenshot of a computer

Description automatically generated

* Intervals, time duration!

A screenshot of a phone

Description automatically generated

A close-up of a number

Description automatically generated

A white background with black text and numbers

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a number

Description automatically generated

A screenshot of a computer program

Description automatically generated

Example code:

-- Select the category and the average completion time by category

SELECT category,

       AVG(date\_completed - date\_created) AS completion\_time

  FROM evanston311

 GROUP BY category

-- Order the results

 ORDER BY completion\_time DESC;

-- Select the current timestamp,

-- and the current timestamp + 5 minutes

SELECT now(), now() + '5 minutes'::interval;

**Date/time components  
 and aggregation**

* Extract component, truncate the value, aggerate the value --> time fields

A white background with black text

Description automatically generated

A screenshot of a phone

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

Example of code:

-- Select name of the day of the week the request was created

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

       -- Select avg time between request creation and completion

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

  FROM evanston311

 -- Group by the name of the day of the week and

 -- integer value of day of week the request was created

 GROUP BY to\_char(date\_created, 'day'), EXTRACT(dow FROM date\_created)

 -- Order by integer value of the day of the week

 -- the request was created

 ORDER BY EXTRACT(dow FROM date\_created);

**Aggregate with Date/time Series**

* The Generate Series only accept **timestamps** (in Python: np.linspace() )

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A blue and white screen with numbers and numbers

Description automatically generated

A screenshot of a computer program

Description automatically generated

A blue screen with white text

Description automatically generated

**Using COALESCE** in a simple SQL query:

**Example Table**

Suppose we have a table called employees with the following data:

| **employee\_id** | **first\_name** | **middle\_name** | **last\_name** |
| --- | --- | --- | --- |
| 1 | John | NULL | Doe |
| 2 | Alice | Marie | NULL |
| 3 | Bob | NULL | Smith |
| 4 | NULL | NULL | Lee |

**Using COALESCE to Get Full Name**

We want to create a new column, preferred\_name, that uses the first available non-NULL name in this order: first\_name, middle\_name, last\_name.

SELECT

employee\_id,

COALESCE(first\_name, middle\_name, last\_name) AS preferred\_name

FROM

employees;

**Result**

| **employee\_id** | **preferred\_name** |
| --- | --- |
| 1 | John |
| 2 | Alice |
| 3 | Bob |
| 4 | Lee |

Comprehensive example:

-- generate series with all days from 2016-01-01 to 2018-06-30

WITH all\_days AS – (subqueries!- First subquery)

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

                             '2018-06-30',

                             '1 day'::interval) AS date),

     -- Subquery to compute daily counts

     daily\_count AS – (Second subquery)

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

             count(\*) AS count

        FROM evanston311

       GROUP BY day)

-- Aggregate daily counts by month using date\_trunc

SELECT date\_trunc('month', date) AS month,

       -- Use coalesce to replace NULL count values with 0

       avg(coalesce(count, 0)) AS average

  FROM all\_days

       LEFT JOIN daily\_count

       -- Joining condition

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

 GROUP BY month

 ORDER BY month;

**Purpose of COALESCE(count, 0)**

In SQL, NULL represents a missing or undefined value. However, NULL values can complicate calculations because most arithmetic operations involving NULL also return NULL. So, COALESCE(count, 0) is used to ensure that any NULL value is treated as 0 instead.

**How COALESCE(count, 0) Works**

1. **If count is NULL**, COALESCE(count, 0) returns 0.
2. **If count has a numeric value**, it returns that value as-is.

**Why It’s Useful in Aggregations**

Without COALESCE(count, 0), any NULL values in count would be excluded in calculations (like SUM, AVG, etc.), which could give inaccurate results. By replacing NULL with 0, you ensure that days without data are considered as having a count of 0 rather than being ignored.

**Example**

Consider this simplified example:

| **day** | **count** |
| --- | --- |
| 2023-01-01 | 5 |
| 2023-01-02 | NULL |
| 2023-01-03 | 3 |

Using COALESCE(count, 0), the values would become:

| **day** | **count** |
| --- | --- |
| 2023-01-01 | 5 |
| 2023-01-02 | 0 |
| 2023-01-03 | 3 |

Now, if you calculate the average count for these days, it will include the day with 0 (instead of skipping it as NULL). This way, all days are accounted for in the calculation.

**Time between events**

* Lead and lag functions

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Code example:

-- Compute the gaps

WITH request\_gaps AS (

        SELECT date\_created,

               -- lead or lag

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

               -- compute gap as date\_created minus lead or lag

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

          FROM evanston311)

-- Select the row with the maximum gap

SELECT \*

  FROM request\_gaps

-- Subquery to select maximum gap from request\_gaps

 WHERE gap = (SELECT MAX(gap)

                FROM request\_gaps);

Code example2:

-- Compute monthly counts of requests created

WITH created AS (

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

              count(\*) AS created\_count

         FROM evanston311

        WHERE category='Rodents- Rats'

        GROUP BY month),

-- Compute monthly counts of requests completed

      completed AS (

       SELECT date\_trunc('month',date\_completed) AS month,

              count(\*) AS completed\_count

         FROM evanston311

        WHERE category='Rodents- Rats'

        GROUP BY month)

-- Join monthly created and completed counts

SELECT created.month,

       created\_count,

       completed\_count

  FROM created

       INNER JOIN completed

       ON created.month=completed.month

 ORDER BY created.month;

A white background with blue text

Description automatically generated