

Introduction to data analysis

Exploratory data analysis and visualisation

Lecture 1: Introduction to SQL for data analysis

Zied Bouyahya

Ecole Centrale de Lyon, Bachelor of Science in data science for responsible business



Useful Resources

- SQLbolt (lessons 1 to 6, and 13 to 18)
- W3School
- Codecademy
- KhanAcademy
- SQLzoo
- TutorialsPoint
- SQL.sh
- SoloLearn
- etc.

Some data sources

- The open data catalog of the government: OpenDataGouv
- The platform: Kaggle
- Relational Dataset Repository

Definition of a Database

- A database is a structured collection of data.
- It allows for:
 - Efficient storage
 - Easy retrieval
 - Management and updating of data
- Example: A library catalog storing information about books.

Types of Databases

- **Relational Databases:**

- Data is organized in tables with rows and columns.
- Examples: MySQL, PostgreSQL.

- **NoSQL Databases:**

- Non-tabular, designed for unstructured or semi-structured data.
- Examples: MongoDB, Firebase.

- **Other Types:**

- Hierarchical databases
- Network databases
- Object-oriented databases

Structure of a Database Table

A table in a relational database consists of rows and columns.

- Each row represents a record or an entity.
- Each column represents a field or attribute of the entity.
- A table typically has a name and is used to store data related to a specific entity (e.g., Customers, Orders).

Primary Key

A primary key uniquely identifies each record in a table.

- It must contain unique values.
- It cannot have NULL values.
- A table can have only one primary key.

Example:

Customer_ID (Primary Key)	Name	Email
---------------------------	------	-------

Foreign Key

A foreign key is a column or set of columns in one table that references the primary key of another table.

- It establishes a relationship between two tables.
- It ensures referential integrity by only allowing values that exist in the referenced table.

Example:

Order_ID (Primary Key) Customer_ID (Foreign Key)

In this example, Customer_ID in the Orders table refers to the Customer_ID in the Customers table.

Example: Primary and Foreign Key in Tables

Consider two tables: Customers and Orders.

Customers Table

Customer_ID	Name	Email
1	Alice	alice@email.com
2	Bob	bob@email.com
3	Charlie	charlie@email.com

Orders Table

Order_ID	Customer_ID	Order_Date
1001	1	2023-01-01
1002	2	2023-01-02
1003	1	2023-01-03

In this case:

- Customer_ID is the primary key in the Customers table.
- Customer_ID in the Orders table is the foreign key that references the primary key in the Customers table.

Relationships Between Tables

Relationships in relational databases define how tables are connected.

There are three main types:

- **One-to-One**
- **One-to-Many**
- **Many-to-Many**

One-to-One Relationship

A one-to-one relationship links one record in a table to one record in another table.

Implementation:

- Use a primary key in one table as a foreign key in the related table.

Users Table

User_ID	Name
1	Alice
2	Bob

Example:

Profiles Table

Profile_ID (PK, FK)	Bio
1	"Likes cats"
2	"Loves programming"

One-to-Many Relationship

A one-to-many relationship links one record in a table to multiple records in another table.

Implementation:

- Use a primary key in one table referenced by a foreign key in another table.

Authors Table

Author_ID	Name
1	Alice
2	Bob

Example:

Books Table

Book_ID	Title	Author_ID (FK)
101	"SQL Basics"	1
102	"Advanced SQL"	1
103	"Data Science"	2

Many-to-Many Relationship

A many-to-many relationship links multiple records in one table to multiple records in another table.

Implementation:

- Use a junction table with foreign keys referencing the primary keys of both tables.

Example:

Students Table

Student_ID	Name
1	Alice
2	Bob

Courses Table

Course_ID	Title
101	Math
102	SQL

StudentCourses (Junction Table)

Student_ID (FK)	Course_ID (FK)
1	101
1	102
2	102

Constraints on Foreign Keys

Foreign keys help maintain referential integrity by enforcing the following constraints:

- **On Delete:** Defines what happens when a record in the referenced table is deleted.
- **On Update:** Defines what happens when the primary key value in the referenced table is updated.
- Options include:
 - CASCADE: Automatically update/delete the foreign key value.
 - SET NULL: Set the foreign key value to NULL.
 - RESTRICT: Prevent deletion or update of the referenced record.

What is SQL?

- SQL stands for **Structured Query Language**.
- It is used to interact with relational databases.
- SQL allows you to:
 - Create databases and tables (CREATE).
 - Insert, update, and delete data (INSERT, UPDATE, DELETE).
 - Retrieve data using queries (SELECT).
- SQL is supported by most relational database management systems (RDBMS), like MySQL, PostgreSQL, and SQLite.

Structure of a SELECT Query

General Syntax

```
SELECT column1, column2, ... FROM table_name WHERE  
condition;
```

- **SELECT**: Specifies the columns to retrieve.
- **FROM**: Specifies the table to retrieve data from.
- **WHERE** (Optional): Filters rows based on conditions.

Example

```
SELECT name, age FROM students WHERE age > 18;
```


Basic Examples of SELECT Queries

- To retrieve all columns from a table:

Query

```
SELECT * FROM employees;
```

- To retrieve specific columns:

Query

```
SELECT first_name, last_name FROM employees;
```

- To rename a column (aliasing):

Query

```
SELECT first_name AS "First Name", last_name AS "Last Name"  
FROM employees;
```

Using Conditions in SELECT Queries

- The **WHERE** clause is used to filter rows.
- Operators:
 - Comparison: =, >, <, >=, <=, <> (not equal)
 - Logical: AND, OR, NOT
 - Pattern matching: LIKE, IN, BETWEEN
- Example:

Query

```
SELECT * FROM employees WHERE department = 'HR' AND salary > 50000;
```

- Retrieves all employees in the HR department earning more than 50,000.

Sorting and Limiting Results

- **ORDER BY:** Sorts the results of a query.
 - Default: Ascending order (ASC).
 - Use DESC for descending order.
- **LIMIT:** Restricts the number of rows returned.
- Example:

Query

```
SELECT first_name, last_name, salary FROM employees  
ORDER BY salary DESC  
LIMIT 5;
```

- Retrieves the top 5 highest-paid employees.

SQL Data Types

- SQL data types define the type of data a column can hold.
- Common categories:
 - **Numeric Data Types:**
 - ▶ INT: Integer values (e.g., 1, 42).
 - ▶ FLOAT, DOUBLE: Decimal numbers.
 - ▶ DECIMAL(p, s): Fixed precision decimal numbers.
 - **Character Data Types:**
 - ▶ CHAR(n): Fixed-length strings (e.g., 'A123').
 - ▶ VARCHAR(n): Variable-length strings (e.g., 'Hello').
 - ▶ TEXT: Long text.
 - **Date and Time Data Types:**
 - ▶ DATE: Stores date (e.g., '2025-01-03').
 - ▶ TIME: Stores time (e.g., '13:45:00').
 - ▶ DATETIME: Stores both date and time.
 - ▶ TIMESTAMP: Stores date and time with time zone.
 - **Other Data Types:**
 - ▶ BOOLEAN: True/False values.
 - ▶ BLOB: Binary data (e.g., images, files).

SQL Date/Time Functions: Converting and Formatting

- SQL provides functions to manipulate and calculate with date and time values.
- **Converting Date and Time:**
 - `STR_TO_DATE(string, format)`: Converts a string into a date.

Example

```
SELECT STR_TO_DATE('01-03-2025', '%d-%m-%Y');
```

- `DATE_FORMAT(date, format)`: Formats a date as a string.

Example

```
SELECT DATE_FORMAT(NOW(), '%Y-%m-%d');
```

- **Current Date and Time:**
 - `NOW()`: Current date and time.
 - `CURDATE()`: Current date.
 - `CURTIME()`: Current time.

SQL Date/Time Functions: Calculating Time Intervals

- SQL provides functions to calculate time intervals.
- **Finding Time Intervals:**
 - DATEDIFF(date1, date2): Difference in days between two dates.

Example

```
SELECT DATEDIFF('2025-01-10', '2025-01-03'); – Returns 7
```

- TIMEDIFF(time1, time2): Difference between two times.

Example

```
SELECT TIMEDIFF('14:30:00', '12:00:00'); – Returns 02:30:00
```

- TIMESTAMPDIFF(unit, datetime1, datetime2): Calculates the difference in specified units (e.g., SECOND, MINUTE, HOUR, DAY, MONTH, YEAR).

Example

```
SELECT TIMESTAMPDIFF(YEAR, '2000-01-01', '2025-01-01'); – Returns 25
```

Using Aggregate Functions

- Aggregate functions perform calculations on multiple rows.
- Common functions:
 - **COUNT**: Counts the number of rows.
 - **SUM**: Calculates the total of a numeric column.
 - **AVG**: Calculates the average value.
 - **MAX**: Finds the highest value.
 - **MIN**: Finds the lowest value.
- Example:

Query

```
SELECT department, AVG(salary) AS avg_salary  
FROM employees  
GROUP BY department;
```

- Calculates the average salary for each department.

Grouping Data with GROUP BY and HAVING

- **GROUP BY:** Groups rows that have the same values in specified columns.
- **HAVING:** Filters groups (similar to WHERE, but used for groups).
- Example:

Query

```
SELECT department, COUNT(*) AS num_employees  
FROM employees  
GROUP BY department  
HAVING num_employees > 10;
```

- Retrieves departments with more than 10 employees.

Question 1: Count flights departing in a specific month

Problem: Count the total number of flights departing in June.

Solution:

```
SELECT COUNT(flight_id) AS total_flights
FROM flights
WHERE MONTH(departure) = 6;
```

Explanation:

- COUNT(flight_id) calculates the total number of flights.
- MONTH(departure) extracts the month from the departure column.
- WHERE MONTH(departure) = 6 filters for flights in June.

Question 2: Average capacity by airline

Problem: Find the average capacity of airplanes for each airline.

Solution:

```
SELECT airline_id, AVG(capacity) AS avg_capacity  
FROM airplanes  
GROUP BY airline_id;
```

Explanation:

- AVG(capacity) computes the average capacity.
- GROUP BY airline_id ensures averages are calculated for each airline.

Question 3: Top 5 most expensive bookings

Problem: List the top 5 most expensive bookings.

Solution:

```
SELECT booking_id, price
FROM bookings
ORDER BY price DESC
LIMIT 5;
```

Explanation:

- ORDER BY price DESC sorts bookings in descending order of price.
- LIMIT 5 restricts the result to the top 5 rows.

Question 4: Total flights per airport

Problem: Count the total number of flights from each airport.

Solution:

```
SELECT departure_airport, COUNT(flight_id) AS total_flights
FROM flights
GROUP BY departure_airport;
```

Explanation:

- COUNT(flight_id) counts flights for each airport.
- GROUP BY departure_airport groups flights by their departure airport.

Question 5: Flights departing after a specific date

Problem: List all flights departing after January 1, 2024.

Solution:

```
SELECT flight_id, departure
FROM flights
WHERE departure > '2024-01-01';
```

Explanation:

- The WHERE `departure > '2024-01-01'` filters flights after January 1, 2024.
- `flight_id` and `departure` are selected for display.

Question 6: Passengers with duplicate bookings

Problem: Identify passengers with more than one booking.

Solution:

```
SELECT passenger_id, COUNT(booking_id) AS total_bookings
FROM bookings
GROUP BY passenger_id
HAVING COUNT(booking_id) > 1;
```

Explanation:

- COUNT(booking_id) counts bookings per passenger.
- HAVING COUNT(booking_id) > 1 filters passengers with multiple bookings.

Question 7: Flights with no bookings

Problem: Find flights that have no bookings.

Solution:

```
SELECT flight_id
FROM flights
WHERE flight_id NOT IN (
    SELECT flight_id
    FROM bookings
);
```

Explanation:

- The subquery `SELECT flight_id FROM bookings` retrieves booked flight IDs.
- `WHERE flight_id NOT IN (...)` filters out flights that are booked.

Question 8: Airline with maximum flights

Problem: Identify the airline with the maximum number of flights.

Solution:

```
SELECT airline_id, COUNT(flight_id) AS total_flights
FROM flights
GROUP BY airline_id
ORDER BY total_flights DESC
LIMIT 1;
```

Explanation:

- COUNT(flight_id) counts flights for each airline.
- ORDER BY total_flights DESC sorts airlines by flight count.
- LIMIT 1 returns only the airline with the highest count.

Question 9: Monthly revenue from bookings

Problem: Calculate the total revenue for each month.

Solution:

```
SELECT MONTH(booking_date) AS booking_month, SUM(price) AS total_revenue
FROM bookings
GROUP BY MONTH(booking_date);
```

Explanation:

- SUM(price) calculates the total revenue.
- GROUP BY MONTH(booking_date) groups bookings by month.

Question 10: Passengers with no bookings

Problem: Find passengers who have no bookings.

Solution:

```
SELECT passenger_id
FROM passengers
WHERE passenger_id NOT IN (
    SELECT passenger_id
    FROM bookings
);
```

Explanation:

- The subquery `SELECT passenger_id FROM bookings` retrieves IDs of passengers with bookings.
- `WHERE passenger_id NOT IN (...)` filters passengers without bookings.

Fundamentals of Computer Programming

Lecture 2 SQL join

Zied Bouyahya

Ecole Centrale de Lyon, Bachelor of Science in data science for responsible business



SQL Joins

- **What's that?**

- Joins in SQL allow combining data from multiple tables in a single query.
- Exploiting the power of relational databases to efficiently obtain results that combine data from different tables.

- **Example**

- Typically involves associating rows from two tables based on the equality of values in a column.
- For instance, a database with "user" and "address" tables can be joined to retrieve user data along with their address in a single query.

- **Another Example**

- Consider a website with tables for "articles" (title, content, publication date, etc.) and "authors" (name, registration date, birth date, etc.).
- Joining these tables allows a single query to display an article along with the author's name, avoiding the need to display the author's name in the "article" table.

SQL Joins

Types of Joins

- **INNER JOIN:** Returns records when the condition is true in both tables. One of the most common joins.
- **CROSS JOIN:** Produces the Cartesian product of two tables, joining each row from one table with every row from another. Results in a large number of records.
- **LEFT JOIN (or LEFT OUTER JOIN):** Returns all records from the left table even if the condition is not met in the other table.
- **RIGHT JOIN (or RIGHT OUTER JOIN):** Returns all records from the right table even if the condition is not met in the other table.
- **FULL JOIN (or FULL OUTER JOIN):** Returns results when the condition is true in at least one of the two tables.
- **SELF JOIN:** Joins a table with itself as if it were another table.
- **NATURAL JOIN:** Natural join between two tables if there is at least one column with the same name in both SQL tables.
- **UNION JOIN:** Union join.

SQL INNER JOIN

Definition

In SQL, the INNER JOIN command, also known as EQUIJOIN, is a common type of join used to link multiple tables. This command returns records when there is at least one row in each table that corresponds to the condition.

Syntax

```
SELECT *  
FROM table1  
INNER JOIN table2 ON table1.id = table2.fk_id
```

The above syntax specifies that you should select records from tables table1 and table2 when the data in the id column of table1 is equal to the data in the column of table2.

SQL INNER JOIN

Alternative Syntax

- The SQL join can also be written as follows:

```
SELECT *  
FROM table1  
INNER JOIN table2  
WHERE table1.id = table2.fk_id
```

- The syntax with the WHERE condition is an alternative way to perform the join but may be less readable if there are already multiple conditions in the WHERE clause.

Example

- Imagine an application with a "user" table and an "order" table containing all orders placed by users.
- To display all orders associated with users, you can use the following query:

```
SELECT id, first_name, last_name, order_date, invoice_number, total_price  
FROM user  
INNER JOIN order ON user.id = order.user_id
```

INNER JOIN

Example

CustomerID	CustomerName	City
1	Carrefour	Tassin
2	Auchan	Bron
3	Monoprix	Villeurbanne

Table: Customers

OrderID	CustomerID	OrderDate
101	1	2024-01-15
102	2	2024-02-05
103	1	2024-03-20

Table: Orders

```
SELECT *  
FROM  
    Orders  
INNER JOIN  
    Customers ON Orders.CustomerID = Customers.CustomerID;
```

OrderID	OrderDate	CustomerID	CustomerName	City
101	2024-01-15	1	Carrefour	Tassin
102	2024-02-05	2	Auchan	Bron
103	2024-03-20	1	Carrefour	Tassin

Table: Result of INNER JOIN

Cross join

- In SQL, the command `CROSS JOIN` is a particular join of two tables that returns the Cartesian product of the two tables.
- Each record in the first table is mapped with each row in the second table.
- Typically, `CROSS JOIN` is combined with a `WHERE` clause to filter the result (the Cartesian product) based on some criteria (or condition).
- For example, if `table_1` contains n records and `table_2` contains m records, the cross join will return an $n \times m$ records.

Syntax

```
SELECT *  
FROM table_1  
CROSS JOIN table_2
```

Cross join

Example

student_id	student_name	gpa
20231998	AW Yerim	3.45
20231345	Naomi Helmbold	4.1
20238843	Wang Deyi	3.9

advisor_id	advisor_name	office
1234	Savinien Jean	12
1235	Vuillemot Romain	16
1236	Bouyahya Zied	17

```
SELECT *  
FROM  
    students  
CROSS JOIN advisors;
```

student_id	student_name	gpa	advisor_id	advisor_name	office
20231998	AW Yerim	3.45	1234	Savinien Jean	12
20231998	AW Yerim	3.45	1235	Vuillemot Romain	16
20231998	AW Yerim	3.45	1236	Bouyahya Zied	17
20231345	Naomi Helmbold	4.1	1234	Savinien Jean	12
20231345	Naomi Helmbold	4.1	1235	Vuillemot Romain	16
20231345	Naomi Helmbold	4.1	1236	Bouyahya Zied	17
20238843	Wang Deyi	3.9	1234	Savinien Jean	12
20238843	Wang Deyi	3.9	1235	Vuillemot Romain	16
20238843	Wang Deyi	3.9	1236	Bouyahya Zied	17

SQL left join

The `LEFT JOIN`, also known as `LEFT OUTER JOIN`, in SQL retrieves all records from the left table, and if there are matching rows in the second (right) table, those are included in the result as well. If there is no match in the right table, `NULL` values are filled for columns from the right table.

Syntax

```
SELECT *  
FROM table1  
LEFT JOIN table2 ON table1.id = table2.fk_id
```

`LEFT JOIN` is particularly useful to retrieve all information from `table1` and the associated information even without matches in the `table2` (in such case, the `NULL` values are used)

SQL left join

Example

emp_id	emp_name	department_id
1	KURAKOVA, Sofia	101
2	XU, Jiani	102
3	MERABET, Camil	103

Table: employees

department_id	department_name
101	IT Department
103	HR Department
104	Marketing

Table: departments

```
SELECT *  
FROM employees  
LEFT JOIN departments ON employees.department_id = departments.department_id;
```

emp_id	emp_name	department_id	department_name
1	KURAKOVA, Sofia	101	IT Department
2	XU, Jiani	102	NULL
3	MERABET, Camil	103	HR Department

LEFT JOIN

Example 2

customer_id	customer_name
1	CEHIC, Mélissa
2	MALIKIREDDY, Reddy
3	HRISHIKESH, Alikatte

Table: Table: customers

order_id	customer_id	order_amount
101	1	150.00
102	2	200.00
103	1	100.00

Table: Table: orders

```
SELECT *  
FROM customers  
LEFT JOIN orders ON customers.customer_id = orders.customer_id;
```

customer_id	customer_name	order_id	order_amount
1	CEHIC, Mélissa	101	150.00
1	CEHIC, Mélissa	103	100.00
2	MALIKIREDDY, Reddy	102	200.00
3	HRISHIKESH, Alikatte	NULL	NULL

Table: Result of LEFT JOIN

RIGHT JOIN

The `RIGHT JOIN`, also known as `RIGHT OUTER JOIN`, in SQL retrieves all records from the right table, and if there are matching rows in the (left) table, those are included in the result as well. If there is no match in the left table, `NULL` values are filled for columns from the left table.

Syntax

```
SELECT *  
FROM table1  
RIGHT JOIN table2 ON table1.id = table2.fk_id
```

RIGHT JOIN Example

Example

customer_id	customer_name
1	DEMODE, Charlotte
2	GONG, Yuxin
3	QI, Haofei

Table: Table: customers

order_id	customer_id	order_amount
101	1	150.00
102	2	200.00
103	1	100.00
104	3	300.00
105	4	120.00

Table: Modified Table: orders

customer_id	customer_name	order_id	order_amount
1	DEMODE, Charlotte	101	150.00
1	DEMODE, Charlotte	103	100.00
2	GONG, Yuxin	102	200.00
3	QI, Haofei	104	300.00
NULL	NULL	105	120.00

Table: Result of RIGHT JOIN

FULL (OUTER) JOIN

FULL JOIN a.k.a. FULL OUTER JOIN is used to combine the results from the two tables based on certain criteria and fills the missing matches with NULL values.

Syntax

```
SELECT *  
FROM table1  
FULL OUTER JOIN table2 on table1.id = table2.fk_id
```


FULL JOIN

Example

customer_id	customer_name
1	BOUYAHYA, Zied
2	MIRONESCU, Elisabeth
3	VUILLEMOT, Romain
4	SAVINIEN, Jean

Table: Table: customers

order_id	customer_id	order_amount
101	1	150.00
102	2	200.00
103	1	100.00
104	3	300.00
105	5	120.00

Table: Modified Table: orders

customer_id	customer_name	order_id	order_amount
1	BOUYAHYA, Zied	101	150.00
1	BOUYAHYA, Zied	103	100.00
2	MIRONESCU, Elisabeth	102	200.00
3	VUILLEMOT, Romain	104	300.00
4	SAVINIEN, Jean	NULL	NULL
5	NULL	105	120.00

Table: Result of FULL JOIN

SELF JOIN

The SELF JOIN command is used to match table information with records from the same table.

Syntax

```
SELECT
`t1`.`name_column1`,
`t1`.`name_column2`,
`t2`.`name_column1`,
`t2`.`name_column2`
FROM `table` as `t1`
JOIN `table` as `t2` ON `t2`.`fk_id` = `t1`.`id`
```

SELF JOIN

Example

id	prenom	nom	email	manager_id
1	Yerim	Aw	aw.yerim@example.com	NULL
2	Jane	Smith	jane.smith@example.com	1
3	Bob	Johnson	bob.johnson@example.com	1
4	Alice	Green	alice.green@example.com	2

SELECT

t1.id, t1.firstname, t1.lastname, t1.email, t1.manager_id,

t2.firstname || ' ' || t2.lastname AS manager_name

FROM employees AS t1

LEFT JOIN employees AS t2 ON t2.id = t1.manager_id;

id	firstname	lastname	email	manager_id	manager_name
1	Yerim	AW	aw.yerim@example.com	NULL	NULL
2	Jane	Smith	jane.smith@example.com	1	Aw Yerim
3	Bob	Johnson	bob.johnson@example.com	1	Aw Yerim
4	Alice	Green	alice.green@example.com	2	Jane Smith

Table: Result of Self-Join