INTRO APP WEB

Politecnico di Torino

Introduzione alle Applicazioni Web

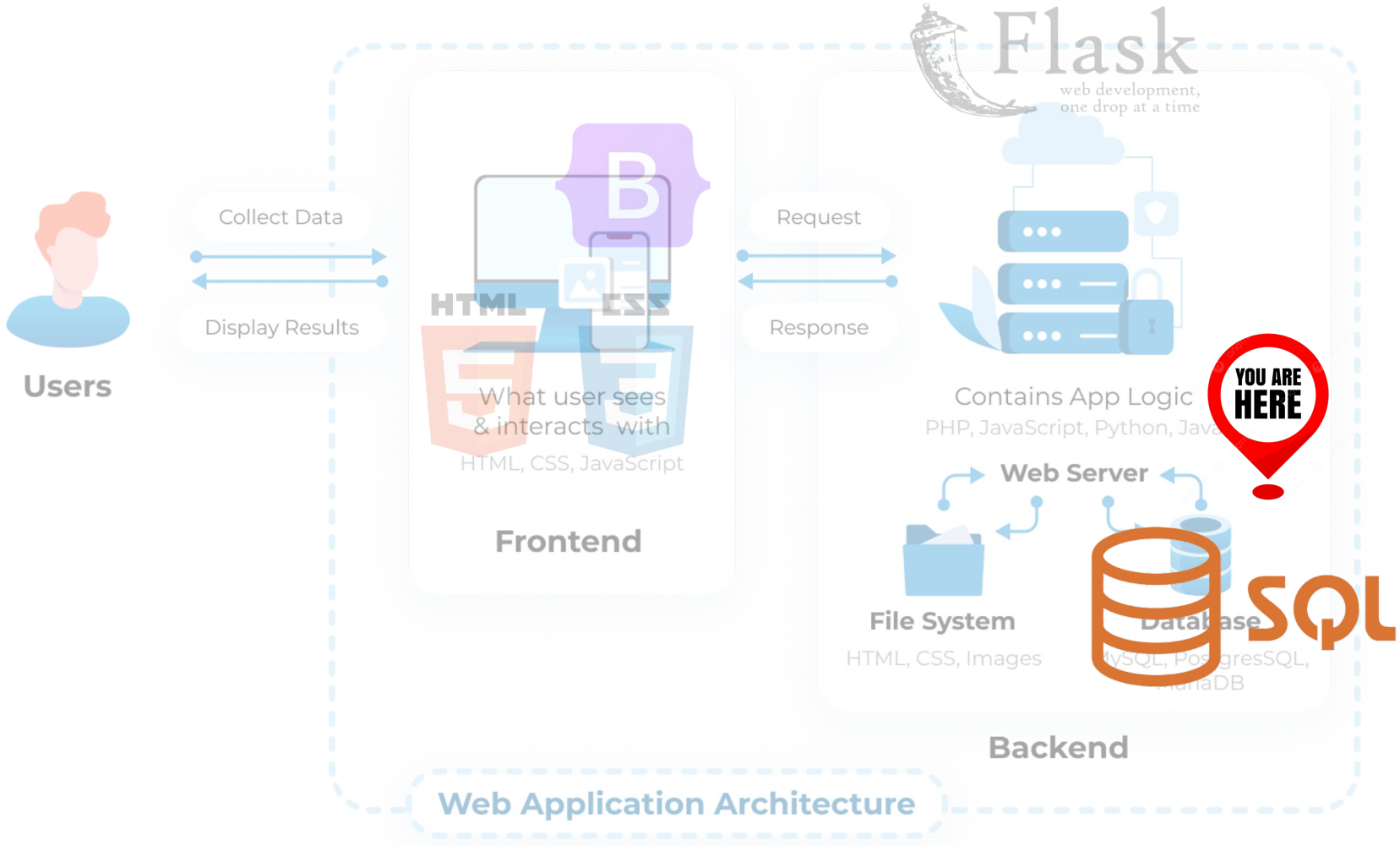
SQL

Juan Pablo Sáenz

Goals

- Understand what **SQL** is and why it is important
- Learn the **basic structure** and **syntax** of SQL commands
- Practice **simple queries** to retrieve, insert, update, and delete data

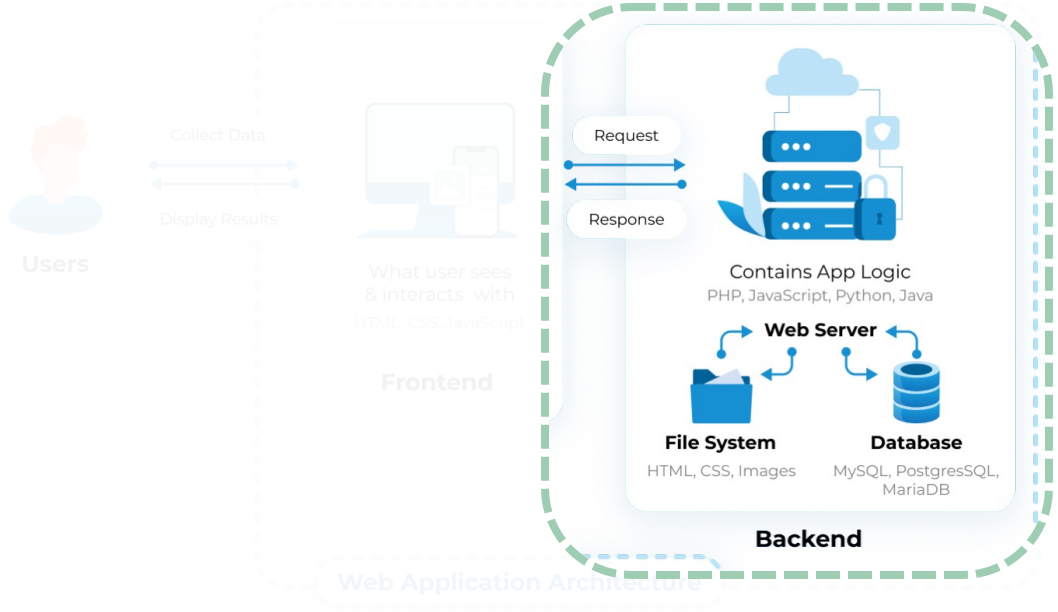
📍 Database: where are we?



Web architecture components: Backend

Database: a system that **stores and organizes data**, making it easy to retrieve, manage, and update.

- It ensures data integrity, security, and performance, often structured in **tables**, **rows**, and **columns**.
- **SQL (Structured Query Language)** is the language used to **interact with databases**, allowing users to search, insert, update, and delete data.

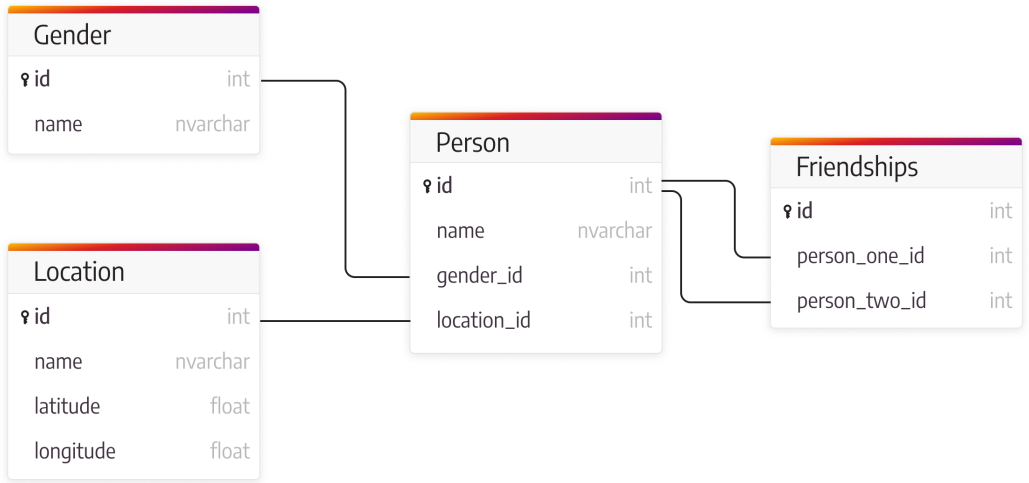


Relational databases

A **structured** way to store data in **tables** (**rows** and **columns**)

Each **table** represents a specific **entity** (e.g., users, products)

- ⚠️ Tables are named in **PLURAL** (e.g., users, orders) to reflect collections

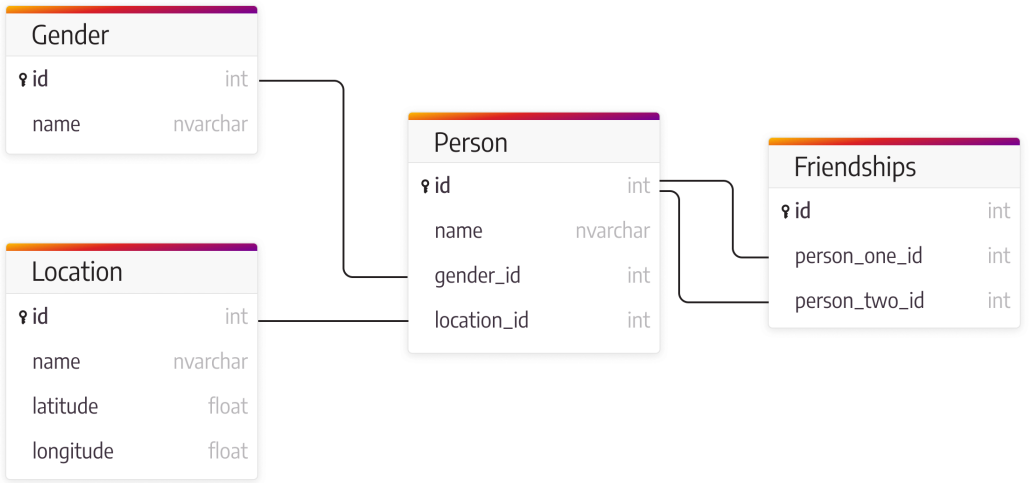


https://memgraph.com/blog/graph-database-vs-relational-database

Relational databases

Tables can be linked through **relationships** (e.g., foreign keys)

Enables **powerful querying using SQL** (Structured Query Language)

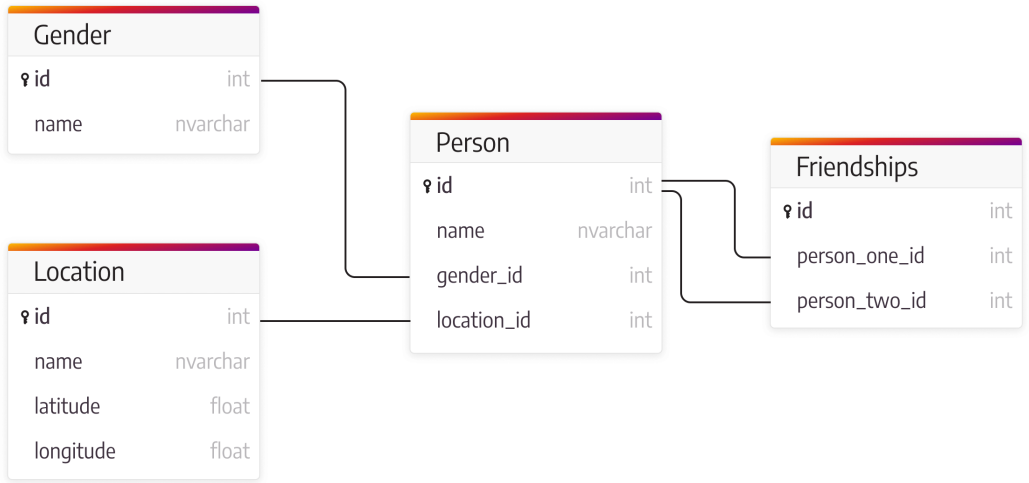


https://memgraph.com/blog/graph-database-vs-relational-database

Relational databases

Ensures data integrity through **ACID** properties:

- **A**tomicity: All or nothing: a transaction must **fully happen or not at all**
- **C**onsistency: **Data must stay correct** before and after a transaction
- **I**solation: **Transactions don't mess with each other**, even if run at the same time
- **D**urability: Once saved, **data won't be lost**—even if the system crashes

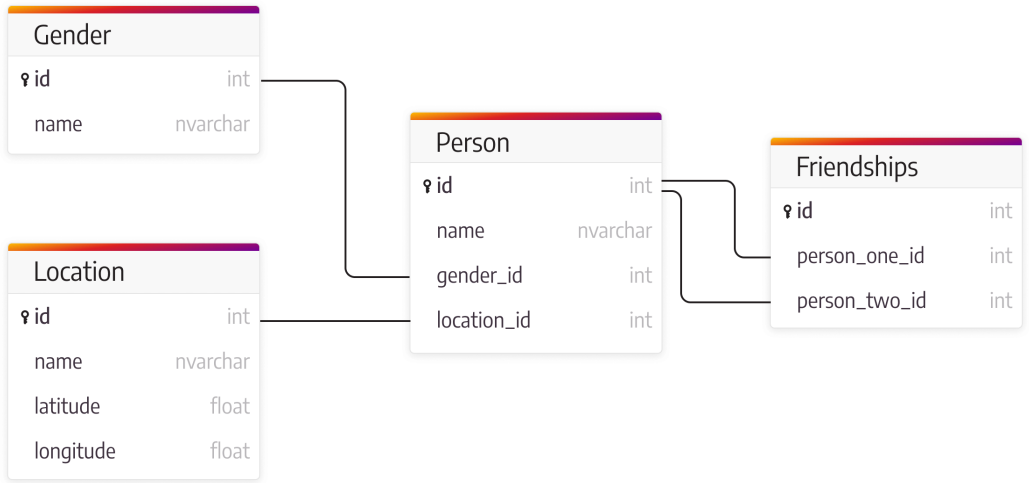


https://memgraph.com/blog/graph-database-vs-relational-database

Relational database keys: PK

🔑 Primary Key (PK)

- Uniquely **identifies** each record in a table
- Must contain **unique** values
- **Cannot** contain **NULL** values

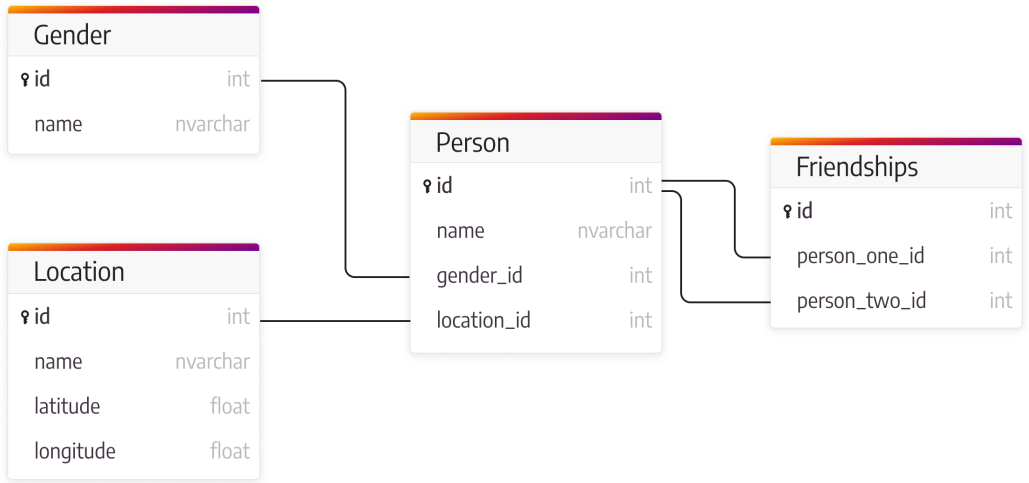


https://memgraph.com/blog/graph-database-vs-relational-database

Relational database keys: PK

💡 **Important considerations**:

- Use a **surrogate key**: a system-generated key, usually an auto-incremented integer
- Use a **single column** for the PK: simpler to manage and index, ensuring better performance
- **Keep It Simple**: A simple primary key is easy to maintain, index, and query



https://memgraph.com/blog/graph-database-vs-relational-database

Relational database keys: FK and UK

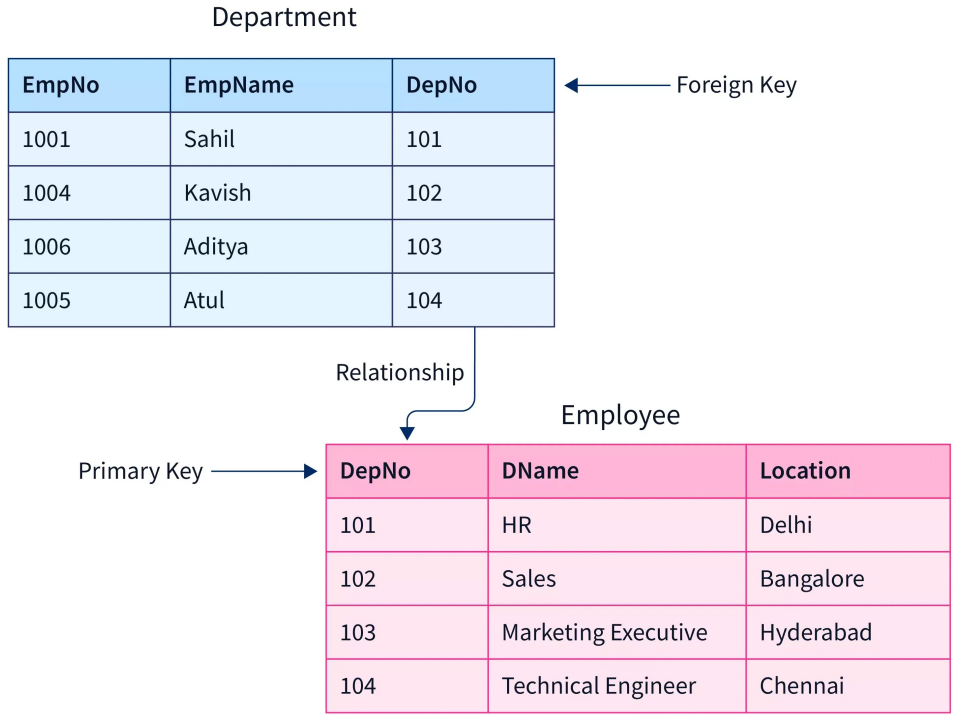
🔑 Foreign Key (FK)

Establishes a **relationship** between two tables

- Linking a column in one table (FK) to the primary key of another

🔑 Unique Key (UK)

- Ensures all values in a column are unique (but can accept NULL values)



https://businessanalytics.substack.com/p/primary-key-vs-foreign-key-in-sql

Relational database: Data types

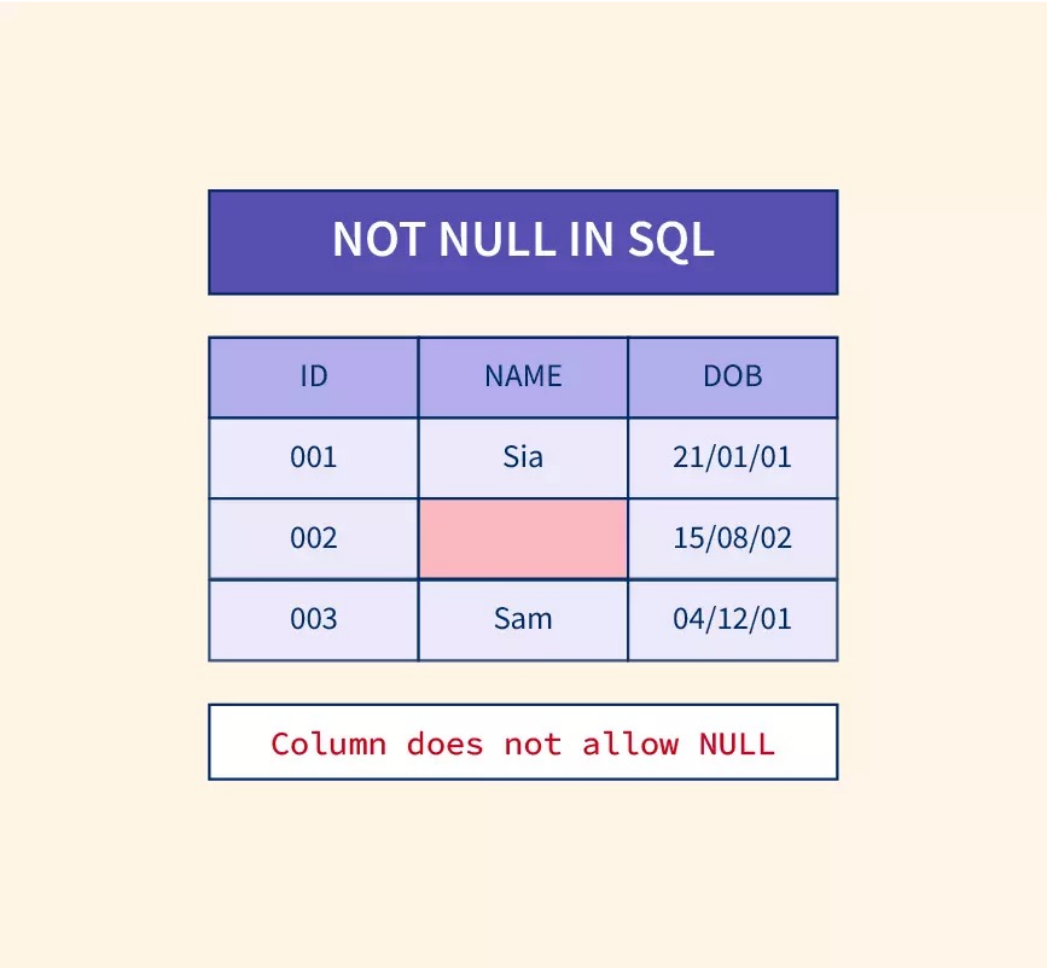
- **INT**: Integer numbers (e.g., employee IDs).
- **VARCHAR**: Variable-length strings (e.g., names, emails).
- **DATE**: Stores date values (e.g., hire date).
- **BOOLEAN**: Stores **TRUE**/**FALSE** values (e.g., is_active).
- **DECIMAL(p, s)**: Fixed-point numbers for financial calculations (e.g., salary).
- **TEXT**: Stores large amounts of text data

⚠️ In SQLite, **REAL** stores floating-point numbers, while **NUMERIC** offers flexibility, allowing storage as integers or real numbers based on the value

Relational database: Not Null Constraint

- Ensures that a column **cannot have NULL values**

- Guarantees that data is always present for that column



https://www.scaler.com/topics/not-null-in-sql/

Relational database: Cardinality

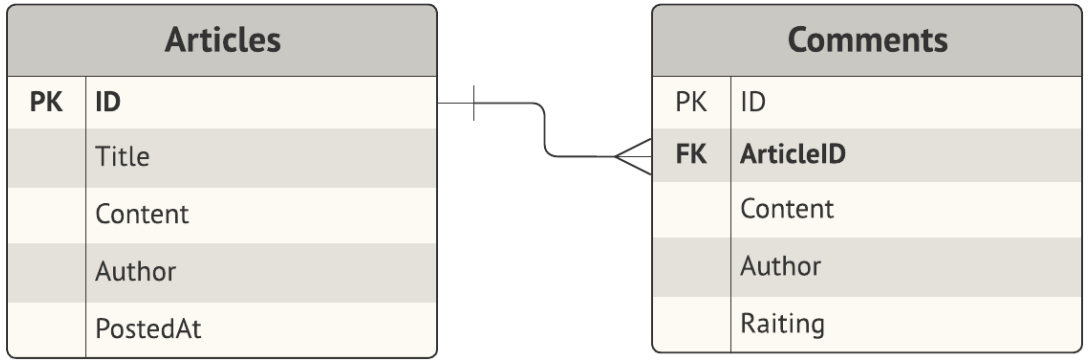
One-to-One (1:1)

- One record in a table corresponds to exactly one record in another table

One-to-Many (1:N)

- One record in a table corresponds to multiple records in another table
- A blog post has many comments

Many-to-One (N:1)

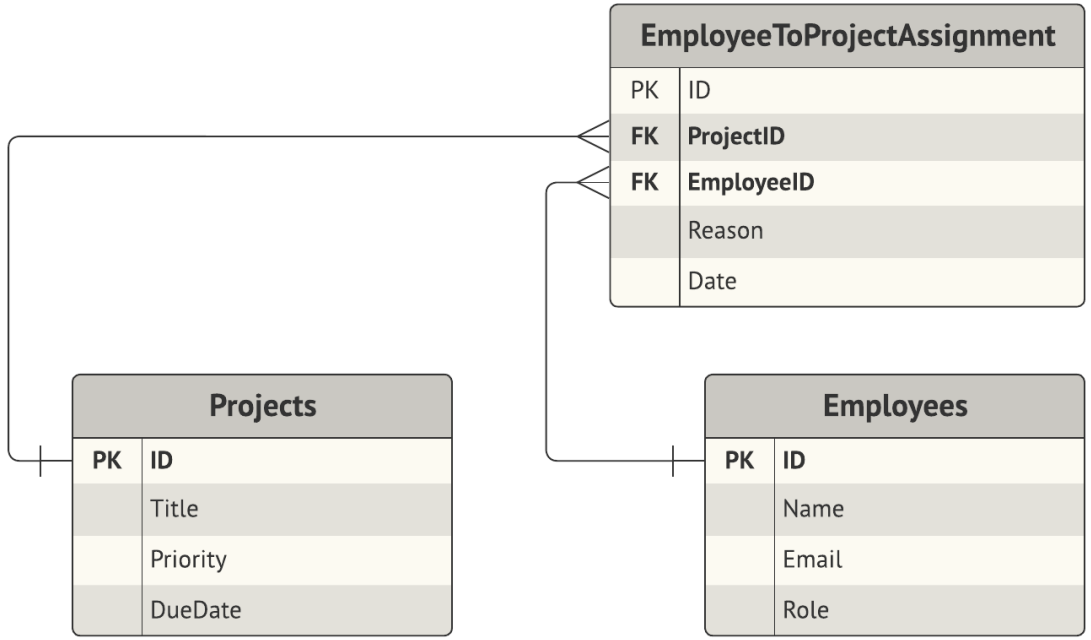


https://levelup.gitconnected.com/many-to-many-one-relationships-are-simple-in-sql-but-hard-in-nosql-2b3cfeeb70eb

Relational database: Cardinality

Many-to-Many (N:M)

- Multiple records in one table correspond to multiple records in another table
- Implemented via a **junction table**
- Students can enroll in **multiple courses**, and each course can have **many students**



https://levelup.gitconnected.com/many-to-many-one-relationships-are-simple-in-sql-but-hard-in-nosql-2b3cfeeb70eb

SQL

A **domain-specific language**, designed for managing and manipulating data in databases

- Unlike **general-purpose** programming languages (like Python, Java, or C++), SQL focuses only on **database operations**

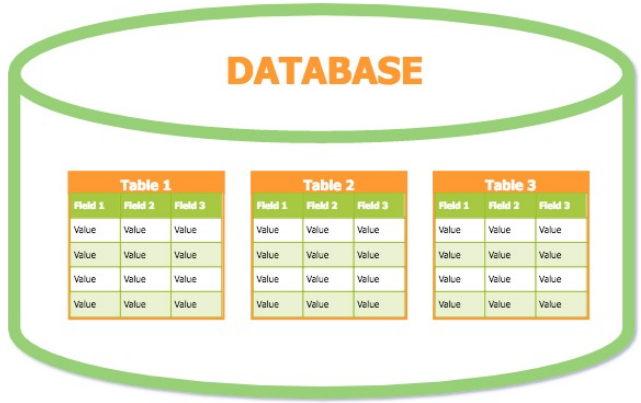
It is **declarative**: you tell the database **what you want**, not how to do it

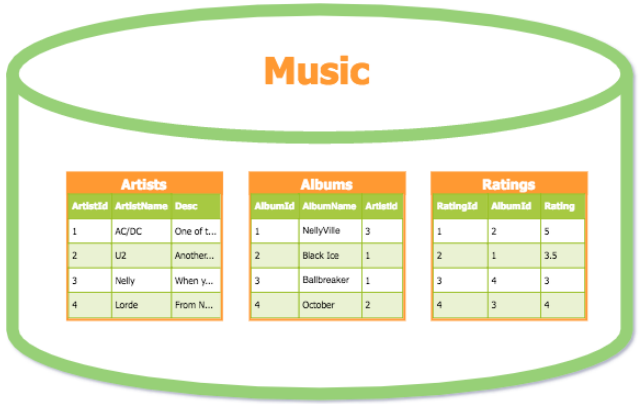
SQL capabilities

Create and **modify tables** and **databases**

Insert, **retrieve**, **update**, and **delete** records

Set **permissions** on database objects





https://database.guide/database-tutorial-part-1-about-databases-creating-databases-tables/

SQL syntax

SQL commands are written in **English-like** statements

Keywords are case-insensitive, but often written in **uppercase**

```sql
SELECT name FROM students
WHERE grade > 90;
```

SQL main commands

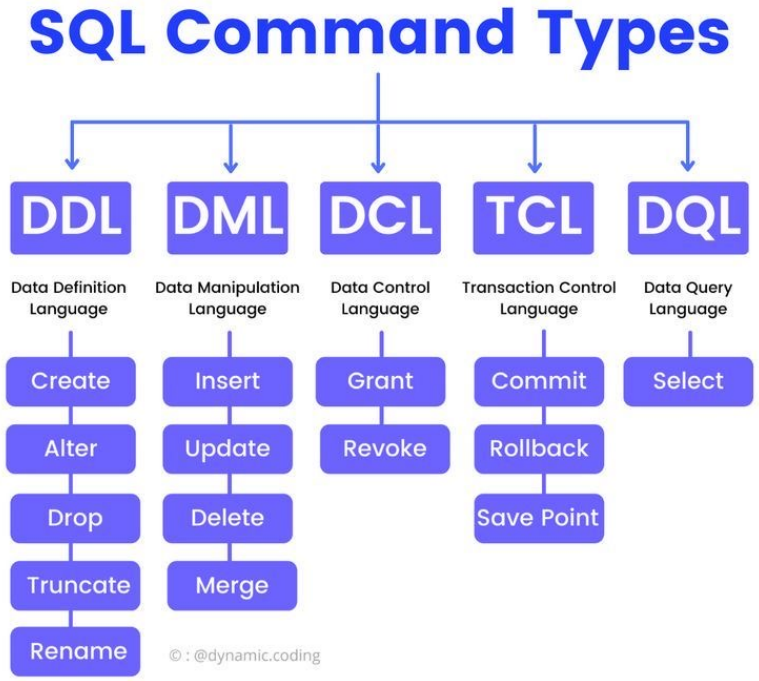
We will focus on Data Manipulation Language (**DML**) and Data Query Language (**DQL**) commands

SELECT: Retrieve data from a table

INSERT: Add new records

UPDATE: Modify existing records

DELETE: Remove records



https://it.pinterest.com/pin/1016898790832478504/

SQL main commands

SELECT: Retrieve data from a table

INSERT: Add new records

UPDATE: Modify existing records

DELETE: Remove records

```sql
SELECT column1, column2, ...
FROM table_name;

SELECT first_name, last_name
FROM employees;
```

SQL main commands

SELECT: Retrieve data from a table

INSERT: Add new records

UPDATE: Modify existing records

DELETE: Remove records

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

SELECT first_name, last_name
FROM employees
WHERE department = 'Sales';
```

SQL main commands

SELECT: Retrieve data from a table

INSERT: Add new records

UPDATE: Modify existing records

DELETE: Remove records

```sql
SELECT column1, column2
FROM table_name
ORDER BY column1 [ASC|DESC];

SELECT first_name, last_name
FROM employees
ORDER BY last_name ASC;
```

SQL main commands

SELECT: Retrieve data from a table

INSERT: Add new records

UPDATE: Modify existing records

DELETE: Remove records

```
SELECT column1
FROM table_name
WHERE condition1 [AND|OR]
condition2;

SELECT first_name, last_name
FROM employees
WHERE department = 'Sales' AND
salary > 50000;
```

SQL main commands

SELECT: Retrieve data from a table

INSERT: Add new records

UPDATE: Modify existing records

DELETE: Remove records

```sql
SELECT column1
FROM table_name
LIMIT number;

SELECT first_name, last_name
FROM employees
LIMIT 5;
```

SQL main commands

SELECT: Retrieve data from a table

INSERT: Add new records

UPDATE: Modify existing records

DELETE: Remove records

```sql
INSERT INTO table_name (column1,
column2, column3, ...)
VALUES (value1, value2, value3, ...);

INSERT INTO Students (ID, Name, Age)
VALUES (1, 'John Doe', 22);

-- If a column is auto-incremented
(like an ID), you may omit it in the
INSERT statement.
```

SQL main commands

SELECT: Retrieve data from a table

INSERT: Add new records

UPDATE: Modify existing records

DELETE: Remove records

```
UPDATE table_name
SET column1 = value1, column2 =
value2, ...
WHERE condition;


UPDATE Students
SET Age = 23
WHERE ID = 1;

-- Without WHERE, all records will
be updated
```

SQL main commands

SELECT: Retrieve data from a table

INSERT: Add new records

UPDATE: Modify existing records

DELETE: Remove records

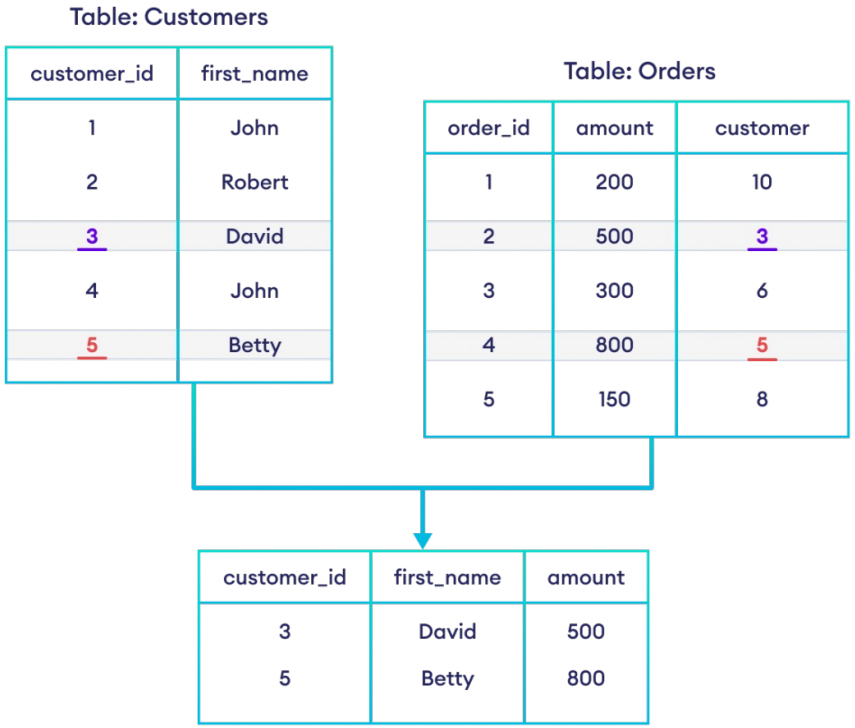
```
DELETE FROM table_name
WHERE condition;

DELETE FROM Students
WHERE ID = 1;

-- Sometimes, it may be better
to disable a row rather than delete
it (e.g., marking it as "inactive")
```

SQL JOIN operations

INNER JOIN: Returns records that have matching values in both tables

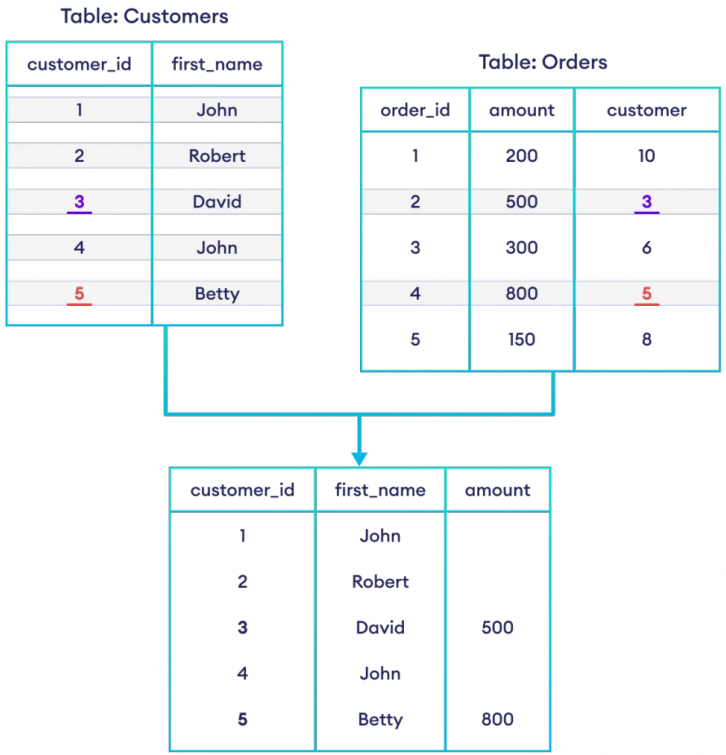


https://www.programiz.com/sql

```sql
SELECT employees.name,
departments.name
FROM employees
INNER JOIN departments ON
employees.department_id =
departments.id;
```

SQL JOIN operations

- **LEFT JOIN (LEFT OUTER JOIN)**: Returns **all records from the left table**, and the matched records from the right table

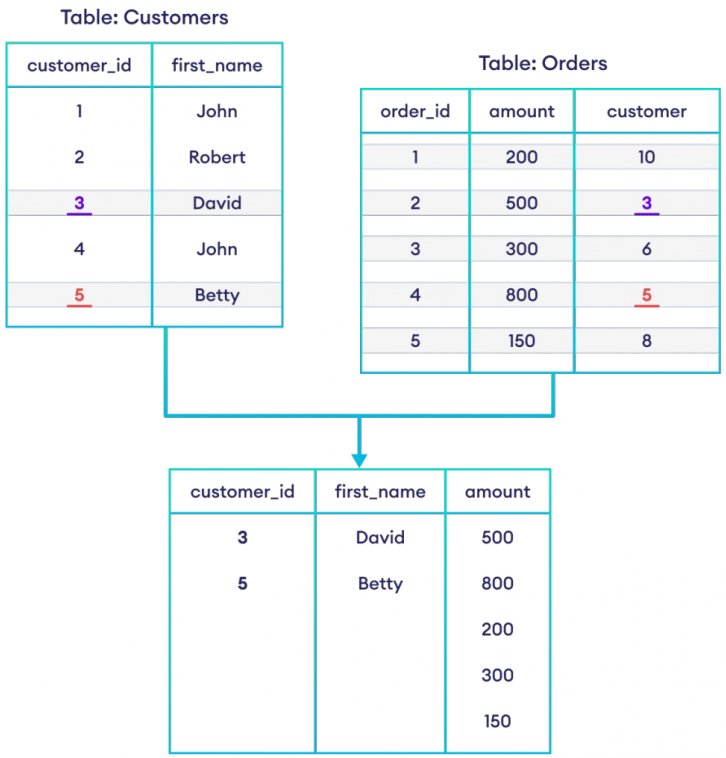


https://www.programiz.com/sql

```
SELECT employees.name,
departments.name
FROM employees
LEFT JOIN departments ON
employees.department_id =
departments.id;
```

SQL JOIN operations

- **RIGHT JOIN (RIGHT OUTER JOIN)**: Returns **all records from the right table**, and the matched records from the left table

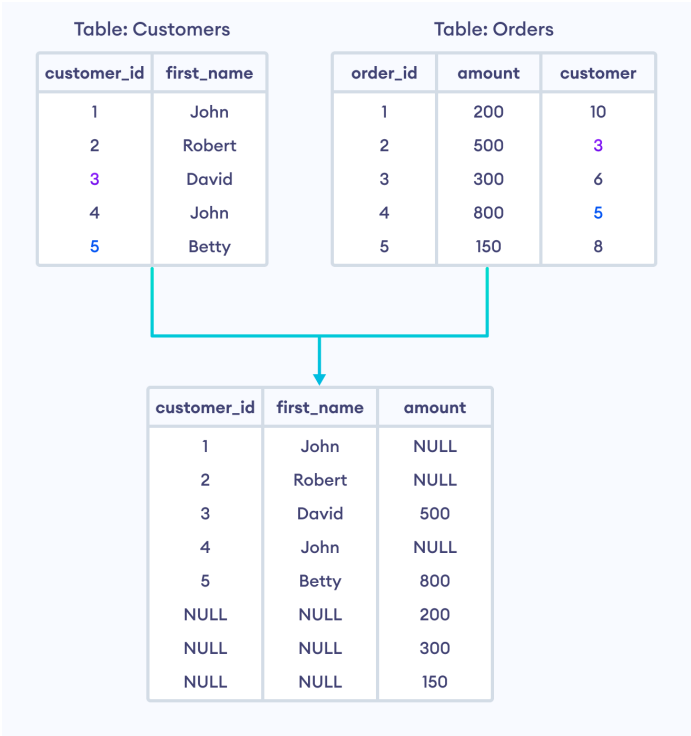


https://www.programiz.com/sql

```sql
SELECT employees.name,
departments.name
FROM employees
RIGHT JOIN departments ON
employees.department_id =
departments.id;
```

SQL JOIN operations

- **FULL JOIN (FULL OUTER JOIN)**: Returns **records when there is a match in either left or right table**



https://www.programiz.com/sql

```sql
-- Combines the results of both LEFT and RIGHT JOINs, returning all employees and all departments, with NULL for unmatched rows

SELECT employees.name, departments.name
FROM employees
FULL OUTER JOIN departments ON employees.department_id = departments.id;
```

Let's see it in practice

Licenza