

SQL for Data Science

Course Outline

1

Lecture 1

1. Database
2. DBMS
3. Relational Vs Transactional model
4. SQL
5. Categories of SQL
8. PostgreSQL Setup
6. DDL
7. Adding Comments

2

Lecture 2

1. SQL Data Types
2. Concept of Keys
3. SQL Commands
4. SQL Constraint
6. Retrieving Data
7. Practice of DDL/DML
8. Creating sample Database for practice.

3

Lecture 3

1. Basic of Filtering
2. Advanced Filtering
3. Wildcards/ Regex
4. Sorting
5. Math operations
6. Aggregate functions
7. Grouping data
8. Combined Practice

4

Lecture 4

1. Using Subqueries
2. Joining Tables
3. Cartesian (Cross) Joins
4. Inner Join
5. Alias and Self Join
6. Advance Joins
7. Full Outer Joins
8. Unions

5

Lecture 5

1. Text String
2. Date-Time String
3. View
4. Data Governance and Profiling
5. Python and SQL
6. Query optimization Tool

What is Database:

A **database** is an organized collection of **data** that is stored and accessed **electronically**. It allows for **efficient storage, retrieval, modification, and management** of data.

- **Data:** Information like names, numbers, dates, etc.
- **Organized:** Stored in tables (like spreadsheets) with rows and columns.
- **Managed:** Through software called a **Database Management System (DBMS)**.
- **Accessed:** Using a language like **SQL (Structured Query Language)**.

Examples: Student records, bank transactions, inventory systems.

What is DBMS?

Database Management System (DBMS) is software for interacting with databases.

- **Functions:** Store, retrieve, update, and delete data.
- **Examples:** PostgreSQL, MySQL, Oracle, SQL Server.
- **Benefits:**
 - Centralized & organized data storage
 - Efficient data access and sharing
 - Ensures consistency and accuracy
 - Manages **multiple users** and **permissions**

Relational DBMS

A **Relational Database Management System** stores data in **tables** (rows & columns) with **relationships** between them using **keys**.

Key Features:

- Structured data with schemas
- Uses SQL for querying
- Data integrity via constraints
- Normalization to remove redundancy

Examples:

- **PostgreSQL** – analytics platforms, GIS systems
- **MySQL** – e-commerce sites (Shopify, WordPress)
- **Oracle DB** – enterprise ERP systems

Transactional DBMS

A **Transactional DBMS** focuses on managing data with **ACID-compliant transactions** — ideal for **mission-critical applications** where **data consistency** is key.

Key Features:

- Supports **Atomicity, Consistency, Isolation, Durability (ACID)**
- Ensures rollback & recovery
- Ideal for high-volume transaction processing

Examples:

- **PostgreSQL** – financial apps with multiple updates
- **IBM Db2** – banking & mainframe systems
- **SQL Server** – retail POS and booking systems

Components of RDBMS

Schema

- A schema is a logical namespace inside a database.
- It groups tables, views, functions, etc.

Example:

```
CREATE SCHEMA hr;
```


Table

- A table is the basic unit of data storage in a database.
- It stores data in rows and columns, like an Excel sheet.

Each row = one record (**Tuple**)

Each column = one field (**Attribute**)

Example:

```
CREATE TABLE hr.employees (  
    id INT PRIMARY KEY,  
    name TEXT,  
    status TEXT  
);
```

View

- A view is a saved SQL query presented like a table.
- It doesn't store data, just shows a filtered or joined result.

Example:

```
CREATE VIEW active_employees AS  
SELECT id, name  
FROM hr.employees  
WHERE status = 'active';
```

What is SQL?

Structured Query Language (SQL)

SQL or Structured Query Language is basically the language that we (the user) use to communicate with the Databases and get our required interpretation of data out of it.

- A standard language for storing, manipulating, and retrieving data in databases.
- Used in all RDBMS (Relational Database Management Systems): PostgreSQL, MySQL, SQLite, SQL Server.

Why SQL in Data Science?

- Retrieve, filter, and transform data
- Prepare datasets for analysis
- Feed data into analytics/ML pipelines

SQL Categories

Category	Full Form	Purpose	Examples
DDL	Data Definition Language	Define/modify DB structure	CREATE, ALTER, DROP
DML	Data Manipulation Language	Manipulate data	SELECT(DQL), INSERT, UPDATE, DELETE
DCL	Data Control Language	Grant/revoke access	GRANT, REVOKE
TCL	Transaction Control Language	Manage transactions	COMMIT, ROLLBACK
DQL	Data Query Language	Query data	SELECT

Installation of PostgreSQL

Shell Command (psql)

1. Get version of PostgreSQL
2. Listing down all database
3. Creating new database
4. Switching to the database
5. Connection Information
6. Some commands related to Tables and Schema

```
SELECT version();
```

```
\l
```

```
CREATE database <db_name>;
```

```
\c <db_name>
```

```
\conninfo
```

```
\dt , \d <table>, \ds , \dv, \di , \dn
```

PGAdmin:

pgAdmin is a **graphical user interface (GUI)** tool for **managing PostgreSQL databases**. It provides a user-friendly way to work with PostgreSQL instead of using the command-line interface (**psql**).

DDL (Data Definition Language)

DDL

DDL (Data Definition Language) is a subset of SQL used to **define, create, modify, and delete** database structures such as tables, schemas, indexes, and views.

Command	Description
<i>CREATE</i>	Creates a new table, database, view, or other object.
<i>ALTER</i>	Modifies an existing object (e.g., add/remove column).
<i>DROP</i>	Deletes an existing object permanently.
<i>TRUNCATE</i>	Removes all records from a table but keeps its structure.
<i>RENAME</i>	Renames a table or column.

Structure of DDL table creation:

```
CREATE TABLE <table_name>(  
    column1 data_type [constraint],  
    column2 data_type [constraint],  
    column3 data_type [constraint],  
    ...,  
    ...,  
    [table_constraints]  
);
```

Example:

Create table

```
CREATE TABLE Students (  
    ID INT PRIMARY KEY,  
    Name VARCHAR(50),  
    Age INT  
);
```

Alter table

```
ALTER TABLE Students ADD  
Email VARCHAR(100);
```

Drop table

```
DROP TABLE Students;
```

Truncate table

```
TRUNCATE TABLE Students;
```

More Example

1. Basic Table with Primary Key and NOT NULL

```
CREATE TABLE Employees (  
    emp_id INT PRIMARY KEY,  
    name VARCHAR(100) NOT NULL,  
    age INT,  
    department VARCHAR(50)  
);
```

2. Table with AUTO INCREMENT / SERIAL

```
CREATE TABLE Customers (  
    customer_id SERIAL PRIMARY KEY,  
    full_name VARCHAR(100) NOT NULL,  
    email VARCHAR(100) UNIQUE  
);
```

4. Composite Primary Key

```
CREATE TABLE Enrollment (  
    student_id INT,  
    course_id INT,  
    enrollment_date DATE,  
    PRIMARY KEY (student_id, course_id)  
);
```

5. Using DEFAULT values

```
CREATE TABLE Products (  
    product_id INT PRIMARY KEY,  
    name VARCHAR(50),  
    quantity INT DEFAULT 0,  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

5. Table with FOREIGN KEY and CHECK constraint

```
CREATE TABLE Orders (  
    order_id INT PRIMARY KEY,  
    customer_id INT,  
    order_date DATE NOT NULL,  
    amount DECIMAL(10, 2) CHECK (amount > 0),  
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)  
);
```

Why we are using ';' at end of the command?

SQL Data Types

Define the kind of data a column can hold.

Category	Data Types	Description
Numeric	INT, FLOAT, DECIMAL	Whole numbers, floating-point, precise nums
Character	CHAR, VARCHAR, TEXT	Fixed and variable-length strings
Date/Time	DATE, TIME, DATETIME	Stores time-based values
Boolean	BOOLEAN	TRUE or FALSE

Explore more data types: [Link](#)

Concept of Keys

Keys **identify each record uniquely** in a table.

- ✓ Uniquely identify records
- ✓ Maintain data integrity
- ✓ Create relationships between tables
- ✓ Optimize queries and indexing
- ✓ Enforce business rules

Explore more about keys: [Link](#)

1. Primary Key:

- Uniquely identifies each row.
- Cannot be NULL or duplicate.

2. Foreign Key:

- Links to primary key in another table.
- Maintains referential integrity.

3. Unique Key:

- Ensures all values in a column are different.
- It can be NULL. (2 NULL values are different from each other)

4. Composite Key:

- Combination of two or more columns to uniquely identify a row.

5. Candidate Key:

- All potential primary keys.

6. Super Key:

- Any column set that uniquely identifies rows.

SQL Constraint

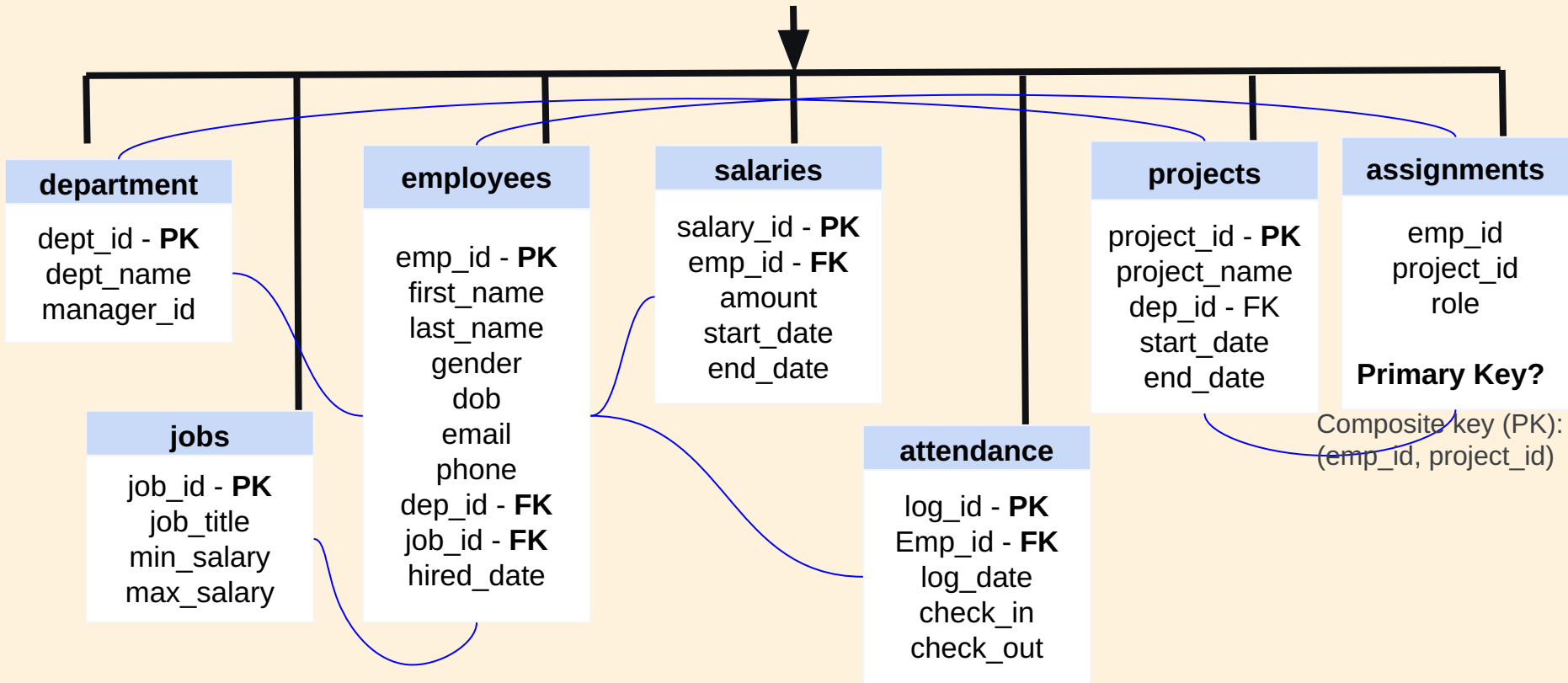
Rules applied to data in a table to maintain accuracy.

Explore more constraint: [Link](#)

Constraint	Description
NOT NULL	Ensures a column cannot have NULL value
UNIQUE	Ensures all values in a column are unique
PRIMARY KEY	Combines NOT NULL and UNIQUE
FOREIGN KEY	Ensures referential integrity
CHECK	Validates data with a condition
DEFAULT	Sets a default value for a column

Database Creation

Human Resources (HR)



Creating New Tables

```
CREATE TABLE <table_name>(  
    column1 data_type [constraint],  
    column2 data_type [constraint],  
    column3 data_type [constraint],  
    ...,  
    ...,  
    [table_constraints]  
);
```

Find pdf for the all the sql queries in resource folder of SQL.

Importing data from csv files

If file contains header and we want to ignore it:

PGAdmin : *COPY <table_name> FROM <file_path> CSV HEADER;*

Psql : *\COPY <table_name> FROM <file_path> CSV HEADER;*

If file doesn't contain any header:

PGAdmin : *COPY <table_name> FROM <file_path> CSV;*

Psql : *\COPY <table_name> FROM <file_path> CSV;*

Basic Data Retrieval queries

Basic Filtering Queries

Structure of Data filtering Query

SELECT [DISTINCT] column1, column2, ...

FROM table_name

[WHERE condition]

[GROUP BY column1, column2, ...]

[HAVING group_condition]

[ORDER BY column1 [ASC|DESC], ...]

[LIMIT number]

[OFFSET number];

Clause / Operator

Example Query Snippet

WHERE

WHERE gender = 'F'

<> (not equal)

WHERE gender <> 'M'

> / <

WHERE amount > 100000

BETWEEN

WHERE amount BETWEEN 50000 AND 100000

IN

WHERE job_id IN ('ENG01', 'HR01')

LIKE

WHERE email LIKE '%@company.com'

IS NULL

WHERE phone IS NULL

NOT IN

WHERE dept_id NOT IN (SELECT dept_id FROM projects)

EXISTS

WHERE EXISTS (SELECT 1 FROM salaries WHERE emp_id = e.emp_id)

Wildcard and Regex

Category	Clause / Operator	Example Query Snippet
Wildcard Matching	LIKE 'A%'	Starts with A → WHERE name LIKE 'A%'
	LIKE '%son'	Ends with "son" → WHERE name LIKE '%son'
	LIKE '%abc%'	Contains "abc" → WHERE email LIKE '%abc%'
	_ (single char)	LIKE '_an' → Matches "Dan", "Jan"
Regex (PostgreSQL)	~ (match)	WHERE username ~ '^[a-z0-9_]+\$' → lowercase letters, numbers, underscores
	~* (case-insensitive)	WHERE name ~* '^a' → Names starting with A or a
	!~ (not match)	WHERE email !~ '^[^@]+@company\.com\$' → not ending with @company.com

Sorting and Limiting

Clause / Operator

Example Query Snippet

ORDER BY

ORDER BY last_name ASC

LIMIT

LIMIT 5

OFFSET

LIMIT 5 OFFSET 10

Math Operation

Clause / Operator	Example Query Snippet
<code>+, -, *, /, %</code>	<code>SELECT 3 + 5 - 2 * 4 / 2 AS result;</code>
Aliasing Calculations	<code>SELECT emp_id, amount * 0.1 AS bonus FROM salaries;</code>
Use in WHERE	<code>WHERE amount + 5000 > 60000</code>
Use in SELECT	<code>SELECT emp_id, amount, amount - 2000 AS adjusted_salary FROM salaries</code>

Aggregation & Grouping

Clause / Operator	Example Query Snippet
COUNT(), SUM()	SELECT dept_id, SUM(amount) FROM salaries GROUP BY dept_id
AVG(), MAX(), MIN()	SELECT AVG(amount), MAX(amount) FROM salaries
GROUP BY	GROUP BY dept_id
HAVING	HAVING COUNT(*) > 5

Subqueries

A **subquery** is a query nested inside another SQL query. It can be used in **SELECT**, **FROM**, or **WHERE** clauses to:

- Filter results
- Compare aggregated values
- Dynamically fetch values

Standard Structure

```
SELECT column1  
FROM table1  
WHERE column2 [operator] (  
    SELECT column2  
    FROM table2  
    WHERE condition  
);
```

Joining Tables

Cartesian Join

Inner Join

Alias and Self Join

Advance Join (Right and Left Join)

Outer Join

Union Operation

String Queries (Data Cleaning)

String Queries (Data Cleaning)

Date-Time Queries (Data Formatting)

View

Data Governance and Profiling

Python and SQL

Python and SQL

Python and SQL

Query Optimization Tool