MODULE 1: CORE PYTHON & DATA

WEEK: 3 LECTURE: 13

DATE: 04/09/2025

INSTRUCTOR: HAMZA SAJID

BEYOND FILES: INTRODUCTION TO SQL & RELATIONAL DATABASES

- Welcome back! We've learned to save our data in files like .txt and .json. This is great for simple applications, but what happens when our data becomes large and interconnected?
- What if we're building a social media app, an e-commerce store, or a blog? Today, we learn the industry-standard solution for managing structured data: Relational Databases and the language we use to talk to them, SQL.

TODAY'S AGENDA

- The "Why" - What is a Database?
  - Analogy: The Digital Filing Cabinet
- Core Concepts of Relational Databases
  - Tables, Rows, Columns, and Keys
  - Example: Designing a simple blog schema
- Exploring the SQL Language
  - The SELECT & FROM statements: For getting data
  - The WHERE & ORDER BY clauses: For filtering and sorting
- Putting It All Together & The Hands-On Lab
  - Hands-On Lab: Querying a Sample Database
  - Q&A and Wrap-up

THE "WHY" - THE PROBLEM WITH FLAT FILES

- Imagine we're running a small online store and we track our orders in a single spreadsheet (or a big JSON file). It might look like this:

OrderID	OrderDate	CustomerName	CustomerEmail	Item	Price
101	2025-09-05	Alice Smith	alice@example.com	T-Shirt	20.00
102	2025-09-05	Bob Johnson	bob@example.com	Coffee Mug	15.00
103	2025-09-06	Alice Smith	alice@example.com	Hat	25.00
104	2025-09-06	Alice Smith	alice.smith@work.com	T-Shirt	20.00

- This seems simple, but it's hiding three major problems.

THE THREE EVILS OF "FLAT" DATA

- Data Duplication: Alice Smith's name and email are repeated for every order she makes. This wastes space and is inefficient.
- Data Inconsistency (Update Anomaly): In Order 104, Alice's email is different! Did she update it? Or was it a typo? Now we have conflicting information. If we need to update her email, we have to hunt down every single row she appears in.
- Data Integrity Issues (Deletion Anomaly): What if Bob Johnson cancels his only order (Order 102)? If we delete that row, we lose all record that Bob was ever a customer!

THE RELATIONAL SOLUTION: SMART TABLES

- A relational database solves this by breaking the data into separate, single-purpose tables.

Customers Table:

CustomerID (PK)	Name	Email
1	Alice Smith	alice@example.com
2	Bob Johnson	bob@example.com

Orders Table:

OrderID (PK)	OrderDate	CustomerID (FK)
101	2025-09-05	1
102	2025-09-05	2
103	2025-09-06	1

- Now, Alice's information exists in only one place. If she updates her email, we change it in one spot, and all her orders are automatically associated with the new information. If Bob's order is deleted, his record in the Customers table remains.

CORE CONCEPTS: WHAT IS A RELATIONAL DATABASE ?

- Imagine a digital filing cabinet designed for perfect organization. A relational database stores data in a structured way, making it easy to find and manage.
- It uses tables that can be linked—or "related"—to each other, which prevents you from having to repeat information.

CORE CONCEPTS: TABLES, ROWS, AND COLUMNS

- Table: A collection of related data, like a spreadsheet. For example, a Customers table.
- Columns: The headers of the table that define the data categories (e.g., CustomerName, City).
- Rows: The individual records in the table. Each row represents a single customer.
- CustomerID CustomerName City
- 1 Alfreds Futterkiste Berlin
- 2 Ana Trujillo México D.F.

CORE CONCEPTS: KEYS

- Keys are special columns that are essential for identifying rows and linking tables together. They are the foundation of the "relational" model.
- There are two main types of keys we will focus on:
  - Primary Key
  - Foreign Key

CORE CONCEPTS: PRIMARY & FOREIGN KEYS

- Primary Key: A column with a unique value for every row. It acts as a unique ID (like a Student ID). It cannot be empty or duplicated. In the Customers table, CustomerID is the Primary Key.
- Foreign Key: A column in one table that refers to the Primary Key of another table. This is how you create a link. If an Orders table has a CustomerID column, that is a Foreign Key linking back to the Customers table.

IN-CLASS EXERCISE 2: IDENTIFYING KEYS

- Look at the two tables below.

Authors Table:

AuthorID	AuthorName	Nationality
1	J.K. Rowling	British
2	George Orwell	British

Books Table:

BookID	Title	AuthorID
101	Harry Potter...	1
102	1984	2
103	Animal Farm	2

What is the Primary Key of the Authors table?
What is the Primary Key of the Books table?
What is the Foreign Key in this schema, and which table is it in?

DATABASE DESIGN: A SIMPLE BLOG SCHEMA

- Let's apply these concepts. We will design a database for a simple blog on paper.
- We need to store information about three things:
  - Users: The people who write posts and comments.
  - Posts: The articles on the blog.
  - Comments: The replies to the posts.

DATABASE DESIGN: THE TABLES

- Users Table
  - UserID (Primary Key)
  - Username
  - Email
- Posts Table
  - PostID (Primary Key)
  - Title
  - UserID (Foreign Key - links to the Users table)
- Comments Table
  - CommentID (Primary Key)
  - CommentText
  - PostID (Foreign Key - links to the Posts table)
  - UserID (Foreign Key - links to the Users table)

DATABASE DESIGN: VISUALIZING THE RELATIONSHIPS

- (A simple diagram showing the three tables with lines connecting the keys)
  - Users (UserID) <--- Posts (UserID)
  - Posts (PostID) <--- Comments (PostID)
  - Users (UserID) <--- Comments (UserID)
- This structure shows that one user can have many posts and many comments, and one post can have many comments.

BASIC SQL QUERIES: “SELECT” AND “FROM”

- These are the two mandatory parts of any query to retrieve data.
- For Example SELECT CustomerName, City FROM Customers;
- SELECT CustomerName, City
  - What it does: Chooses the columns you want to see.
- FROM Customers;
  - What it does: Specifies which table to look in.
- This query reads as: "From the Customers table, select and show me the CustomerName and City columns."

BASIC SQL QUERIES: “WHERE”

- The WHERE clause filters your rows based on a specific condition. It always comes after FROM.
- SELECT * FROM Customers WHERE City = 'London';
  - What it does: Sets a condition. It will only return rows where the value in the City column is exactly 'London'.
- This query reads as: "From the Customers table, show me all columns, but only for the rows where the city is 'London'."

BASIC SQL QUERIES: “ORDER BY”

- The ORDER BY clause sorts your final results. It always comes after FROM and WHERE.
- SELECT CustomerName, City FROM Customers ORDER BY CustomerName;
  - What it does: Sorts the returned rows. By default, it sorts in alphabetical or numerical ascending (ASC) order. You can use DESC for descending order.
- This query reads as: "From the Customers table, show me the CustomerName and City, and sort the entire result list alphabetically by the customer's name."

HANDS-ON LAB: QUERYING A SAMPLE DB

- Goal:
  - Use an online SQL playground to run basic queries against a sample database.
- Tool:
  - We will use SQLite Online, which is free, requires no installation, and comes with a pre-loaded sample database.

STEP 1: OPEN THE SQL PLAYGROUND

- Open your web browser and go to sqliteonline.com.
- The website will load a demo database automatically.
- On the left side, you can see the table names (e.g., Customers, Products).
- The top-middle pane is the SQL Editor where you will type your queries.
- The bottom-middle pane is where the Results will appear after you run a query.

STEP 2: RUN A “SELECT” QUERY

- Task: Get the name and price of every product.
- In the SQL Editor, type the following query:
  - SELECT ProductName, Price FROM Products;
- Explanation: This query tells the database: "Look inside the Products table, and for every row, show me only the ProductName and Price columns."
- Click the "Run" button. You should see a two-column list in the Results pane.

STEP 3: RUN A “WHERE” QUERY

- Task: Find all customers who are from the country "Germany".
- Delete the old query and type this new one in the SQL Editor:
  - SELECT CustomerName, City FROM Customers WHERE Country = 'Germany';
- Explanation: This query says: "From the Customers table, show me the CustomerName and City, but only for the rows where the Country column contains the text 'Germany'."
- Click "Run". The results should only show customers from Germany.

RUN AN “ORDER BY” QUERY

- Task: Get the names of all suppliers, sorted alphabetically.
- Delete the old query and type this new one in the SQL Editor:
  - SELECT SupplierName FROM Suppliers ORDER BY SupplierName;
- Explanation: This query says: "From the `Suppliers` table, show me the `SupplierName` for all suppliers, and sort the final list alphabetically."
- Click "Run". You will see a single, alphabetized column of supplier names.

BONUS CHALLENGE

- Challenge 1: Descending Order
  - Modify your last query to list the suppliers in reverse alphabetical order (from Z to A).
  - Hint: The ORDER BY clause has a keyword for descending order.
- Challenge 2: Pattern Matching
  - Write a new query to find any product whose name contains the word "Chef".
  - Hint: You can't use = for this. Search for the SQL LIKE operator and the % wildcard character.