🚀 TechSaksham Training - Day 7 Notes

🔹 Virtual Environments in Python

Virtual environments allow us to create isolated Python environments where project-specific dependencies can be installed without interfering with system-wide packages. This is especially useful for working on multiple projects that require different package versions.

Why Use Virtual Environments?

- **Avoid dependency conflicts** – Different projects may require different versions of the same package.
- **Keep the global environment clean** – Installing packages system-wide can cause conflicts.
- **Improve project portability** – Ensures the same package versions are used in different setups.
- **Essential for data analytics, machine learning, and web development.**

Commands to Work with Virtual Environments

Installing `virtualenv` (if not already installed)
```
pip install virtualenv
```

1.

Creating a Virtual Environment
```
python -m venv myenv  # Creates a new environment named 'myenv'
```

2.
3. **Activating the Virtual Environment**

Windows:
```
.\myenv\Scripts\activate
```

    -

Mac/Linux:
```
source myenv/bin/activate
```

    -

Deactivating the Virtual Environment
deactivate

4.

Checking Installed Modules in the Environment
pip freeze

5.

Installing Specific Packages
pip install requests

6.

Viewing Package Details
pip show requests

7.

🔹 Working with the `datetime` Module in Python

The `datetime` module provides various functions to work with date and time operations efficiently.

Operations Performed:

Importing Required Modules
import datetime

-

Getting the Current Date and Time
now = datetime.datetime.now()

print("Current Date and Time:", now)

-

Extracting Year, Month, and Day
print("Year:", now.year)

print("Month:", now.month)

print("Day:", now.day)

-

Getting Only the Current Date
today = datetime.date.today()

print("Today's Date:", today)

-

**Formatting the Date using **`strftime()`
formatted_date = now.strftime("%d-%m-%Y %H:%M:%S")

print("Formatted Date:", formatted_date)

-

Calculating a Future Date (e.g., E-commerce Delivery Estimation)
future_date = today + datetime.timedelta(days=7)

print("Estimated Delivery Date:", future_date)

-

🔹 Introduction to MySQL

What is MySQL?

- **MySQL is a relational database management system (RDBMS)** used to store, retrieve, and manage structured data.
- **Ensures efficient data organization, security, and integrity.**
- **Commonly used in web applications, finance, and e-commerce.**

🔹 MySQL Data Types

1️⃣ Numeric Data Types

- `BIGINT`, `INT`, `SMALLINT`, `TINYINT` (for storing numerical values)

2️⃣ String Data Types

- `CHAR()`, `VARCHAR()`, `TEXT` (for storing textual data)

3️⃣ Boolean Data Type

- `TRUE`, `FALSE` (for logical operations)

4️⃣ Date & Time Data Types

- `DATETIME`, `DATE` (for tracking time-based events)

🔹 MySQL Constraints

Constraints ensure data integrity by restricting invalid entries.

Constraint	Description	Can be NULL?
PRIMARY KEY	Uniquely identifies each row	❌ No
UNIQUE	Ensures unique values	✅ Yes
FOREIGN KEY	Ensures referential integrity	✅ Yes
NOT NULL	Prevents NULL values	❌ No
CHECK	Restricts values based on condition	✅ Yes
DEFAULT	Assigns default value	✅ Yes

AUTO_INCREMENT	Generates unique numeric values	❌ No

📂 Database Commands

```
SHOW DATABASES;

CREATE DATABASE database_name;

USE database_name;
```

🏗️ Creating Tables

```
CREATE TABLE table_name (

column1 datatype constraints,

column2 datatype constraints,

...

[table_constraints]

);
```

📋 Query Examples

```
SHOW TABLES; -- List all tables in database

DESC table_name; -- Show table structure

ALTER TABLE student ADD COLUMN Id INT PRIMARY KEY FIRST;

ALTER TABLE student MODIFY age INT CHECK(age<=20);

INSERT INTO studentdata(Id, first_name, age) VALUES (11, 'Hey', 16);

UPDATE studentdata SET last_name="hello" WHERE Id = 11;

SELECT * FROM employee WHERE name LIKE 'A%';

SELECT MAX(salary) FROM employee;

SELECT salary FROM employee ORDER BY salary DESC LIMIT 2,1;
```

🔗 MySQL Joins

Right Join: Returns all rows from the right table and matching rows from the left.

Left Join: Returns all rows from the left table and matching rows from the right.

Outer Join: Returns all rows from both tables, whether they match or not.

Inner Join: Returns only matching rows from both tables.

Full Outer Join: Done using UNION operator.

Self Join: A table joins itself.

Natural Join: Automatically joins tables using common columns.

🔹 E-Commerce Database Case Study

💡 Features:

- **Cart** (Connected to user and product tables)
- **Product and User Tables** (Independent)
- **Order** (Connected to user, product, cart, and payment)
- **Payment** (Connected to product and user via `uid`)
- **Address** (Connected to user and order)

📌 SQL Queries for E-Commerce Case Study

SELECT * FROM user INNER JOIN cart ON user.uid = cart.uid;

SELECT user.uid, user.uname, product.ppname, product.pprice FROM user

INNER JOIN cart ON user.uid = cart.uid

INNER JOIN product ON cart.pid = product.pid;

SELECT * FROM user

INNER JOIN payment ON user.uid = payment.uid

INNER JOIN product ON product.pid = payment.pid

INNER JOIN orderr ON orderr.payid = payment.payid

INNER JOIN shaddress ON orderr.aid = shaddress.aid;

🔹 Conclusion

At the end of the session, we successfully understood:

- **Python Virtual Environments** and their significance.
- **Date & Time manipulation using `datetime` module.**
- **MySQL database design, constraints, and query execution.**
- **E-Commerce database architecture with real-world case studies.**
- **Practical applications of SQL joins and constraints.**

📜 **Looking forward to applying this knowledge in future projects!** 🚀