(3:58:20)

- To work with Postgres Database within Python application, we need Postgres drivers. This is actually a single or a couple of Python libraries.

- We will use Psycopg for this purpose. It is a PostgreSQL database adapter for Python

- Install the library using following command :-

```
[ pip install psycopg2
```

- For its basic usage functions and syntaxes, we can go to the official documentation for Psycopg2, and then open Basic Module Usage.

Establishing Connection :-

- Just like any other adapter (like mysql-connector), we need to first make a connection object, and using it, we will get a cursor object. Using the cursor object, we will execute SQL queries, and at last commit the changes, and close the connection.

- Using the connect() method of psycopg2, we establish connection to database. It takes keyword arguments :-

host, database, user, password, through which we provide details to connect to specific database.

This library is a bit different, in the sense that when we fetch data, it returns the values only (in a tuple), without the column names.
So, it is a bit difficult to know which data belongs to which attribute, for any record.

To get the values alongwith the column names, we need to import and use another module named **RealDictCursor**

```
from psycopg2.extras import RealDictCursor
```

Now, the code to establish connection, & get the objects:-

```
import psycopg2
try:
    conn = psycopg2.connect(host='localhost',
    database='fastapi', user='postgres',
    password='Ultimategg', cursor_factory=RealDictCursor)

    cursor = conn.cursor()

    print("Connection established successfully")
except Exception as error:
    print("Database Connection failed! Error:", error)
```

database name of our app

IP / domain in which database is running

this will give the column names

⇒ The RealDictCursor must be passed as the cursor_factory keyword argument to the connect() method

⇒ In course of execution of connect() & cursor() method, if any error occurs, then it will be of Exception class, which can be handled accordingly

For development & debugging, we store the Exception in a variable, & print it, to see the error message.

⇒ We want that if connection establishment fails, then it should keep trying again and again, till connection happens.

For this, we can apply a loop, such that only upon successfull connection, the loop ends. Else, the try to establish connection will happen continuously (after some delay) if it fails:-

```
import time
while True:
    try:
        conn = psycopg.connect(....)
        ---
        break
    except Exception as error:
        ---
        time.sleep(2)
```

→ on success, the loop will break.

→ after error, program will sleep for 2 seconds, and then the loop will run from beginning

> Now, we can execute SQL queries using the cursor object's execute() method.

① Path operation for getting all the entries / all posts :-

```
@app.get("/posts")
async def get_posts():
    query = """ SELECT * FROM posts; """
    cursor_object.execute(query)
    all_posts = cursor_object.fetchall()
    return {"data": all_posts}
```

> When we execute any SELECT query, the result that is fetched gets stored in the cursor object itself.

To get all the records in the result, we need to call the fetchall() method with that cursor object.

```
cur.execute(""" SELECT ..... """)
obj = cur.fetchall()
```

The returned results are now stored in an object obj. Now, the results (data from database) can be processed in Python, or returned as result.

> Normally, the fetchall() function returns all the records as a list of tuples, with no column names. But, as we have set cursor_factory = RealDictCursor the records returned will be in the form of list

of RealDictCursor objects.

② Creating a Post Path Operation :-

```python
# Creating a post
@app.post("/posts", status_code = status.HTTP_201_CREATED)
async def create_post(post: Post):
    query = f" INSERT INTO posts (title,content,published) VALUES (%s,%s,%s) RETURNING *; "
    cur.execute(query,(post.title,post.content,post.published))
    created_post = cur.fetchone()
    # After the post being created, saving changes to database
    conn.commit()
    if created_post==None:
        raise HTTPException(status_code = status.HTTP_406_NOT_ACCEPTABLE, detail = "Due to some
    else:
        return {"detail": "Post created successfully!","data": created_post}
```

The 2nd parameter must be an iterable

≥ In the query string, we are not using f-string method, to place the obtained data into SQL query. Instead, we are using %s placeholder in place of data in SQL query, and passing the data as 2nd parameter of execute() method, as a tuple.

This is done to prevent SQL injection. The data passed by user as string may contain arbitary SQL queries. So, passing id directly is not safe. When passed using %s placeholder, and as 2nd parameter in execute() function, the string are cleaned and appropriately filtered.

≥ After making any changes to database, to get it saved, we must commit the changes using commit() function.

commit() method is called using the Connection Object, that is obtained by establishing database connection.

Till being committed, all changes remain as Staged (in Git), so, does not reflect in database.

③ Deleting a Post : Path Operation :-

```
# Deleting a post
@app.delete("/posts/{id}", status_code = status.HTTP_204_NO_CONTENT)
async def delete_post(id: int):
    query = """ DELETE FROM posts WHERE id=%s RETURNING *; """
    cur.execute(query,(id,))
    deleted_post = cur.fetchone()
    conn.commit()
    if deleted_post==None:
        raise HTTPException(status_code = status.HTTP_404_NOT_FOUND, detail = f"Post with id={id} do
    else:
        return Response(status_code = status.HTTP_204_NO_CONTENT)
```

⇒ Simple SQL statement for deleting record is used.

⇒ After deleting, we must apply the RETURNING keyword, so that by checking the deleted post being returned, we know whether post existed or not.

⇒ For successfull deletion, status code should be 204 - No content.

④ Updating a Post : Path Operation :-

⇒ Simply, UPDATE SET command can be used for record deletion.