Week 7: Application Development

L7.1: *Application Design and Development/1: Architecture*

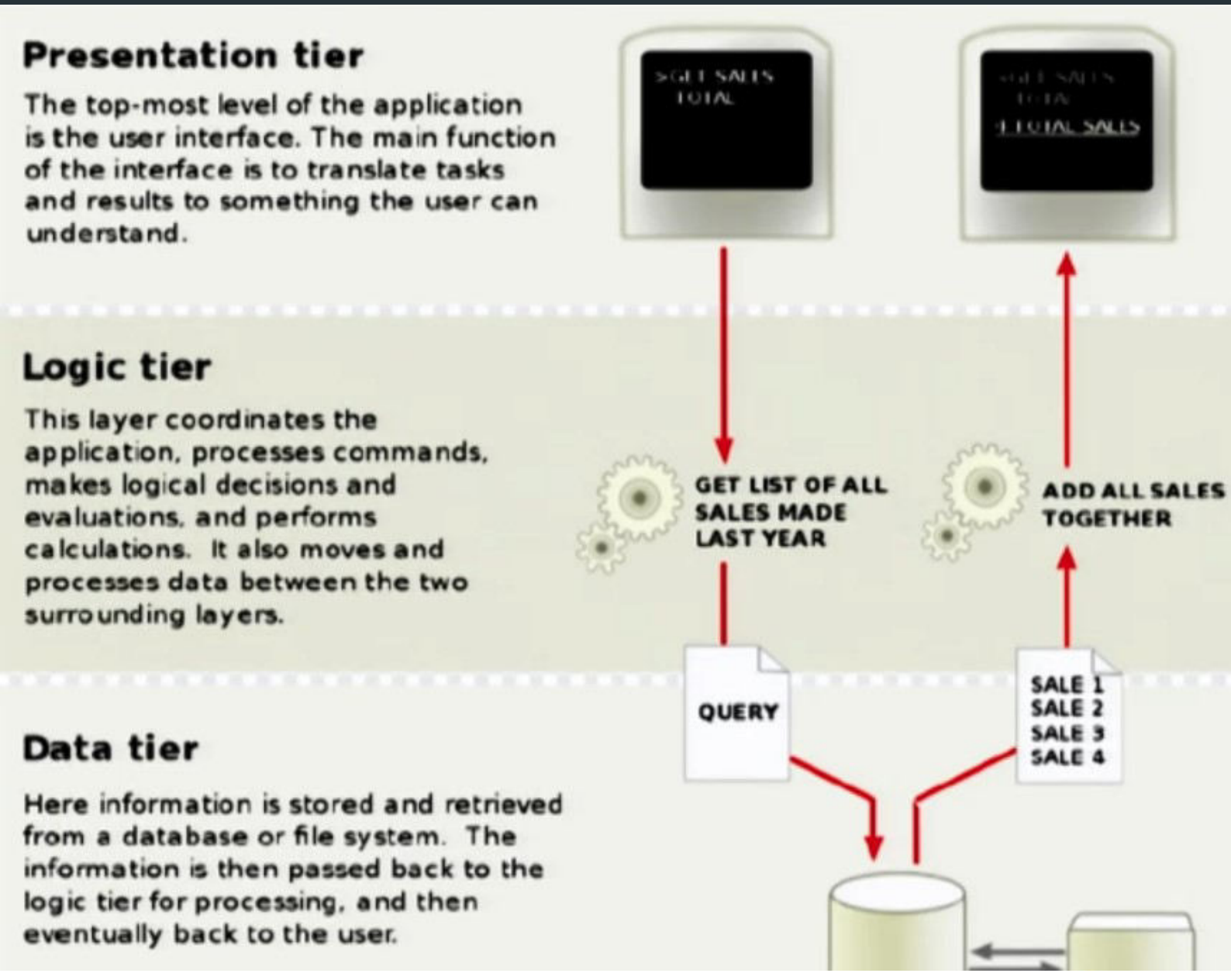
Application Programs and Architectures

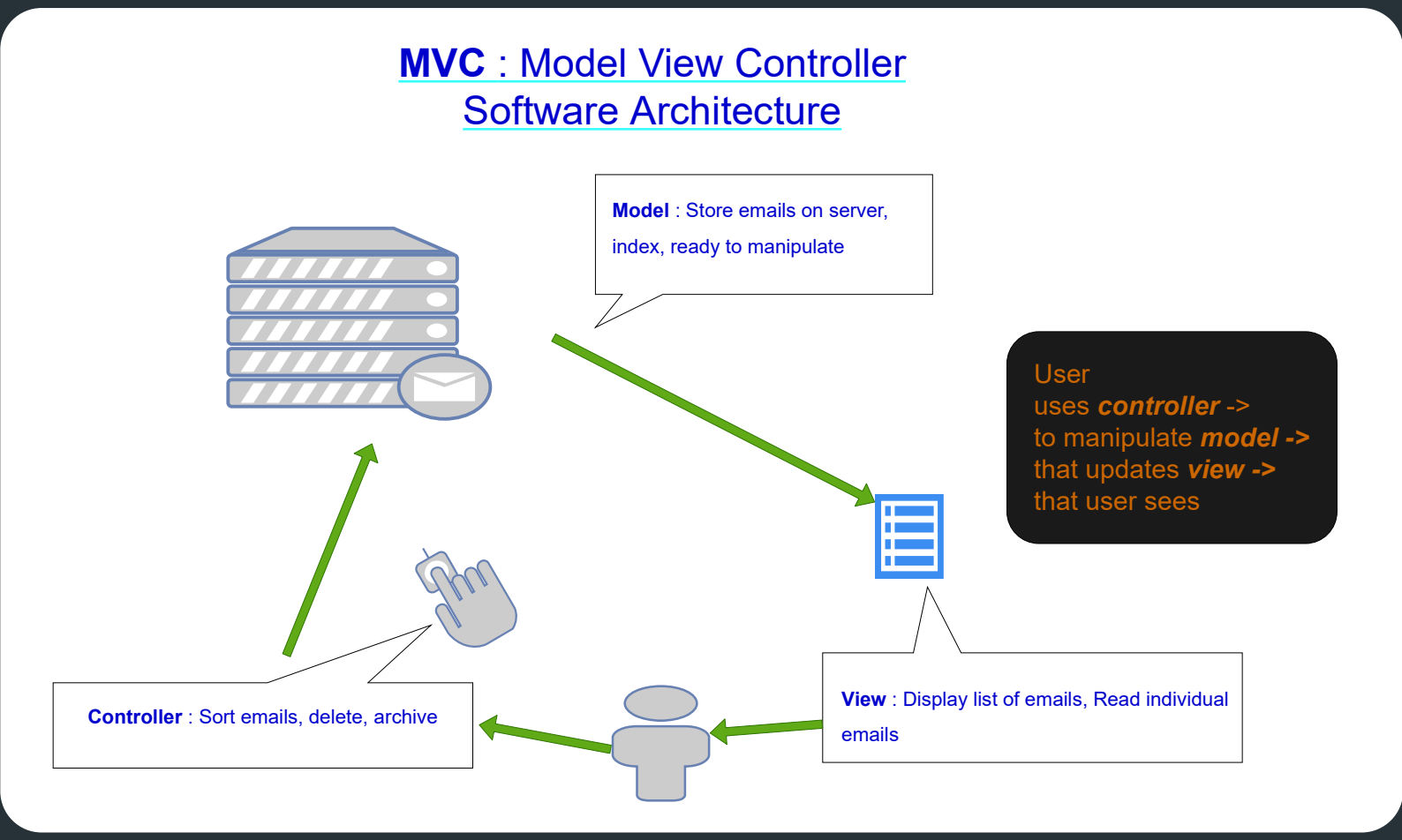
Application Programs

Characterstics

- **Diversity** - There are a large variety of application programs used in the world today in various industries, whether its' financial, health, education, library, travel & tourism, communication, knowledge discovery etc...
- **Unity** - Most aplications use an RDBMS like Oracle, DB2, MySQL etc... for managing data.
    - Applications are functionally split into frontend, middle and backend layers.

Architecture





Learn about Architecture and MVC here 🔗

Object Relational Mapping

- **Object Relational Mapping (ORM)** is a programming technique for converting data between relational databases and object oriented programming languages such as Java, C++, Python etc...
- Schema designer has to provide a mapping between the object model and the relational schema.
  - Example: A class called `Student` with attributes `name`, `age`, `rollno` etc... can be mapped to a table called `Student` with columns `name`, `age`, `rollno` etc...
  - An object can map to multiple tuples in multiple tables.
- Application opens a session, which connects to the database.
- Objects can be created and saved to the database using `session.save(object)`.
- Query can be run to retrieve objects satisfying specified predicates.

Architecture Classification

- Database architecture uses programming languages to design a particular type of software for business or organizations.
- Database architecture focuses on the design, development, implementation and maintenance of computer programs that store and organize information for business, agencies and institutions.
- A database architect develops and implements software to meet the needs of users.

- The design of a DBMS depends on its architecture, It can be *centralized* or *decentralized* or *hierarchical*.
- The architecture of a DBMS can be seen as either single tier or multi-tier.
    - **1-tier architecture** - It is the simplest architecture, where the user directly interacts with the database.
    - **2-tier architecture** - It is the client-server architecture, where the user directly interacts with the application and the application interacts with the database.
    - **3-tier architecture** - It is the web-based architecture, where the user interacts with the web browser and the web browser interacts with the web server and the web server interacts with the database.
    - **n-tier architecture** - It is the distributed architecture, where the user interacts with the web browser and the web browser interacts with the web server and the web server interacts with the application server and the application server interacts with the database server.

Sample applications in multiple tiers

Application	Presentation	Logic	Data	Functionality

Web Mail	• Login
	• Mail List View
	  ◦ Inbox
	  ◦ Sent Items
	  ◦ Outbox
	  ◦ Trash
	• Mail Composer
	• Filters
		• User Authentication
		• Connection to Mail Server (SMTP, POP, IMAP)
		• Encryption / Decryption
			• Mail Users
			• Address Book
			• Mail Items
				• Send / Receive Mails
				• Manage Address Book

Net Banking	• Login • Account View • Add / Delete Account • Add / Delete Beneficiary • Fund Transfer	• User Authentication • Beneficiary Authentication • Transaction Validation • Connection to Banks / Gateways • Encryption / Decryption	• Account Holders • Beneficiaries • Accounts • Debit / Credit Transactions	• Check Balance and Transactions • Transfer Funds

Timetable	• Login
	• Add / Delete Courses, Teachers, Rooms, Slots
	• Assignments:
	  ◦ Teachers → Course
	• Allocations
	  ◦ Course → Room, Slots
	• Views
		• User Authentication
		• Timetable Assignment Logic
		• Encryption / Decryption
			• Courses
			• Teachers
			• Rooms
			• Slots
			• Assignments
			• Allocations
				• Manage timetable for multiple courses taken by multiple teachers

---

L7.2: *Application Design and Development/2: Web Applications*

You can refer this video 🔗

You can refer to these notes 🔗

L7.3: *Application Design and Development/3: SQL and*

Native Language

Working with SQL and Native language

- Applications use **Application Programming Interface** (API) to interact with a database server.
- Applications make calls to
    - Connect to the database server
    - Execute SQL queries
    - Fetch results
    - Close the connection

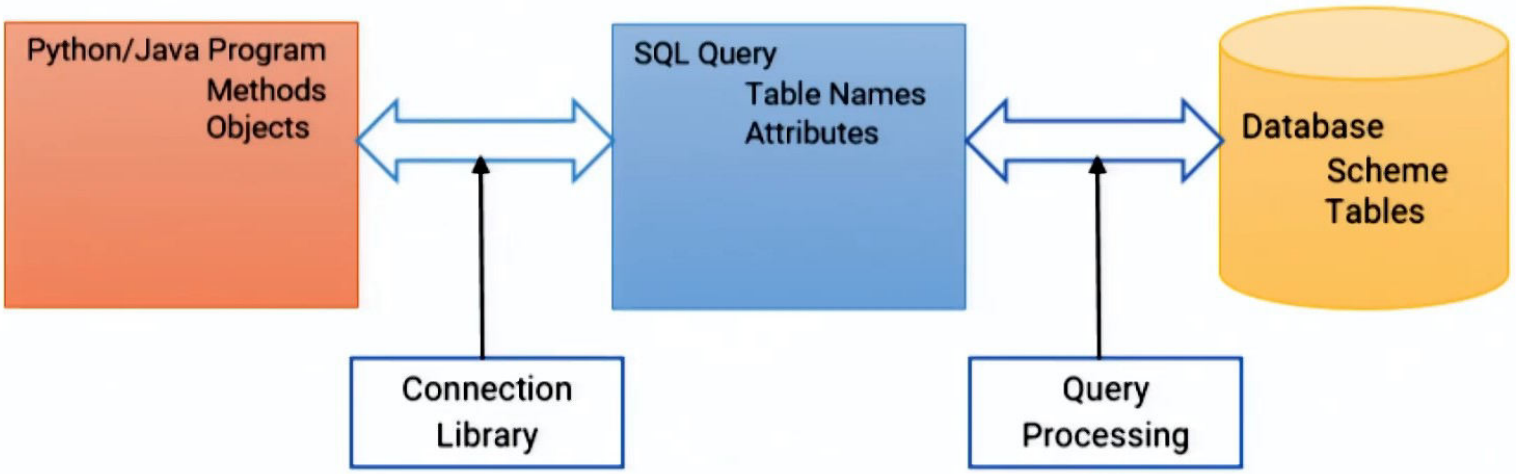
Frameworks

Connectionist

- ***Open Database Connectivity (ODBC)*** works iwth C, C++, C#, Visual Basic, Python etc...
- Other APIs include OLEDB, ADO.NET etc...
- ***Java Database Connectivity (JDBC)*** works with Java.

Embedding

- ***Embedding SQL*** works with C, C++, Java, COBOL etc...



ODBC

- ODBC is a standard API for accessing databases.
- It is independent of the database and the programming language.
- An application written using ODBC can be ported to other platforms, both on the client and server side with few changes to the data access code.
- Applications such as GUI, Spreadsheets etc... can use ODBC.

Example

Reading the data

```
import pyodbc

# Connect to the database
conn = pyodbc.connect('DSN=mydb;UID=myuser;PWD=mypassword')

# Create a cursor
cursor = conn.cursor()

# Execute SQL query
cursor.execute("SELECT * FROM Employees")

# Fetch and display results
for row in cursor.fetchall():
    print(row)

# Close the connection
conn.close()
```

Writing the data

```
import pyodbc

# Connect to the database
conn = pyodbc.connect('DSN=mydb;UID=myuser;PWD=mypassword')

# Create a cursor
cursor = conn.cursor()

# Insert data
insert_query = "INSERT INTO Employees (FirstName, LastName, Department, Salary) VALUES (?, ?, ?, ?)"
data_to_insert = ('John', 'Doe', 'IT', 60000)
cursor.execute(insert_query, data_to_insert)

# Commit the transaction
conn.commit()

# Close the connection
conn.close()
```

JDBC

- ***Java Database Connectivity (JDBC)*** is a standard API for accessing databases from Java.
- It is a Java-based data access technology used for Java database connectivity.
- JDBC supports a variety of features for querying and updating data and for retrieving query results, metadata retrieval, such as querying about relations present in the database and the names and types of relation attributes.
- Model for communicating with the database:

- Open a connection
- Create a "statement" object
- Execute queries using the Statement object to send queries and fetch results
- Exception mechanism to handle errors

Example

- In this example, the code connects to a database, prepares an SQL insert query, binds parameter values, and executes the query to add new data to the "Employees" table. The data consists of the first name, last name, department, and salary of an employee. After executing the query, the transaction is committed, and the resources are closed.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class JdbcInsertExample {
    public static void main(String[] args) {
        String jdbcUrl = "jdbc:mysql://localhost:3306/mydb";
        String user = "myuser";
        String password = "mypassword";

        try {
            // Establish connection
            Connection connection = DriverManager.getConnection(jdbcUrl, user, password);

            // Create prepared statement
            String insertQuery = "INSERT INTO Employees \
            (FirstName, LastName, Department, Salary) VALUES (?, ?, ?, ?)";
            PreparedStatement preparedStatement = connection.prepareStatement(insertQuery);

            // Set parameters
            preparedStatement.setString(1, "Jane");
            preparedStatement.setString(2, "Smith");
            preparedStatement.setString(3, "HR");
            preparedStatement.setInt(4, 55000);

            // Execute update
            int rowsAffected = preparedStatement.executeUpdate();
            System.out.println("Rows inserted: " + rowsAffected);

            // Close resources
            preparedStatement.close();
            connection.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Bridge

A Bridge is a mechanism that allows data exchange between different database systems and programming languages.

- A Bridge is a special kind of a driver that uses another driver-based technology.
- This driver translates *source-function-calls* into *target-function-calls*.
  Some common bridges are:

ODBC-JDBC Bridge

- This bridge allows Java applications using JDBC to communicate with databases that provide ODBC drivers.
- Example: OpenLink ODBC-JDBC Bridge, SequeLink Bridge etc...

JDBC-ODBC Bridge

- The Sequelink JDBC-ODBC Bridge allows JDBC-based applications to access databases with ODBC drivers.
- Example: Sequelink JDBC-ODBC Bridge

OLE DB to ODBC Bridge

- This bridge enables applications using OLE DB to communicate with data sources using ODBC drivers.
- Example: Microsoft OLE DB Provider for ODBC Drivers

ADO.NET to ODBC Bridge

- The System.Data.Odbc namespace in .NET Framework enables ADO .NET applications to communicate with ODBC drivers.
- Example: .NET Framework's System.Data.Odbc Namespace

Embedded SQL

- The SQL standard defines embedding of SQL in a variety of programming languages such as C, C++, Java.
- A language to which SQL queries are embedded is referred to as a **host language**.
- The basic form of these languages follows that of the System R embedding of SQL into PL/1.
- `EXEC-SQL` is used to indicate the beginning of an SQL statement.
- Before executing any SQL statements, the program must first connect to the database.
  `EXEC-SQL connect to server user user-name usign password;`
- Variables in the host language can be used in SQL statements by prefixing them with a colon withing `DECLARE` section:

```
EXEC-SQL BEGIN DECLARE SECTION
    int salary;
EXEC-SQL END DECLARE SECTION
```

```
EXEC-SQL select * from employees where salary > :salary;
```

L7.4: *Application Design and Development/4: Python and PostgreSQL*

Working with PostgreSQL and Python

- There are a lot of modules in Python that can be used to connect to a PostgreSQL database.
- Example: `psycopg2`, `pg8000`, `py-postgresql`, `SQLAlchemy` etc...

> We will be using `psycopg2` module to connect to a PostgreSQL database.
> You can read about it here 🔗

`psycopg2` module

- Its an external module, we need to install it.
- We will use `pip` package manager to install:
  - Windows

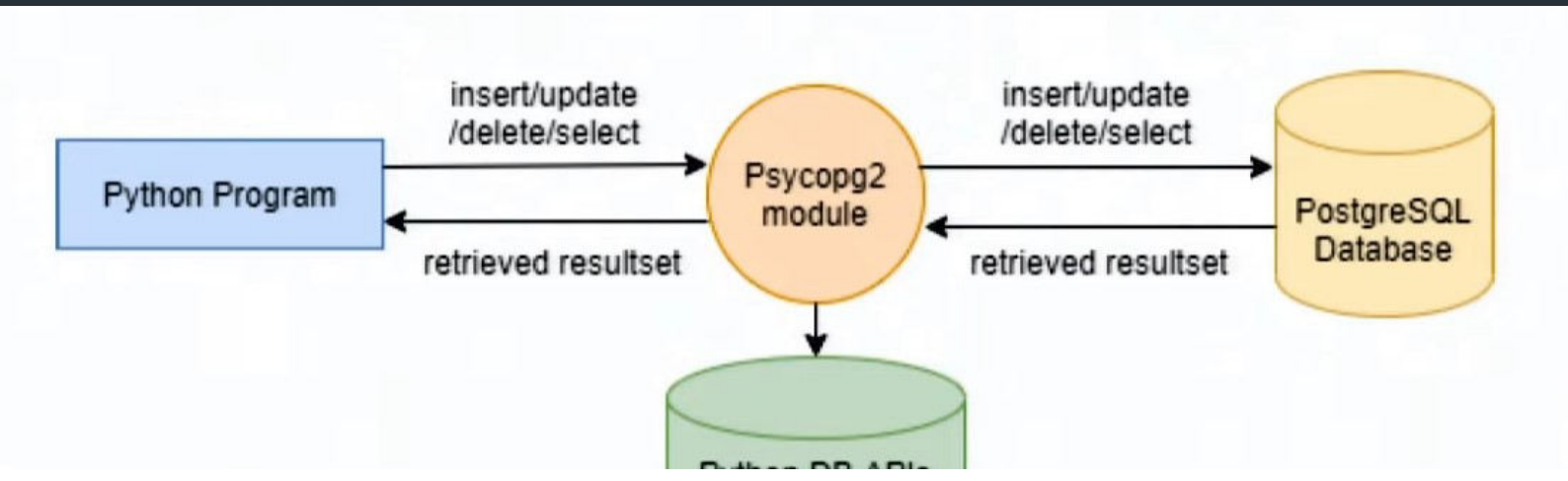
```
pip install psycopg2
```

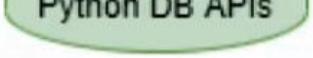
  - MacOS/Linux

```
pip3 install psycopg2
```

Steps to access PostgreSQL from Python

1. Create connection
2. Create cursor
3. Execute the query
4. Fetch / commit / rollback
5. Close cursor
6. Close connection





1. Create connection

```
import psycopg2

connection = psycopg2.connect(
    database="database_name",
    user="user_name",
    password="password",
    host="127.0.0.1",
    port="3737"
)
```

- This will create a connection object which can be used to execute queries.

2. Create cursor

```
cursor = connection.cursor()
```

- This will create a cursor object which can be used to execute queries.

3. Execute the query

```
cursor.execute("SELECT * FROM employees")
```

- This will execute the query and store the result in the cursor object.

4. Fetch / commit / rollback

```
cursor.fetchall()
```

- Here we are fetching the results, because we have executed a `SELECT` query.
- For `INSERT`, `UPDATE`, `DELETE` queries, we need to commit the changes.
- For `ROLLBACK`, we need to rollback the changes.

5. Close cursor

```
cursor.close()
```

- This will close the cursor object.

6. Close connection

```
connection.close()
```

- This will close the connection object.

> **Note**: There are a lot of other methods available in the `psycopg2` module, you can read about them here 🔗
> We can use `psycopg2` module in backend for handling database operations of any application, whether its a web application or a desktop application.

L7.5: *Application Design and Development/5: Application Development and Modbile* 🔗

- You can watch the lecture here 🔗