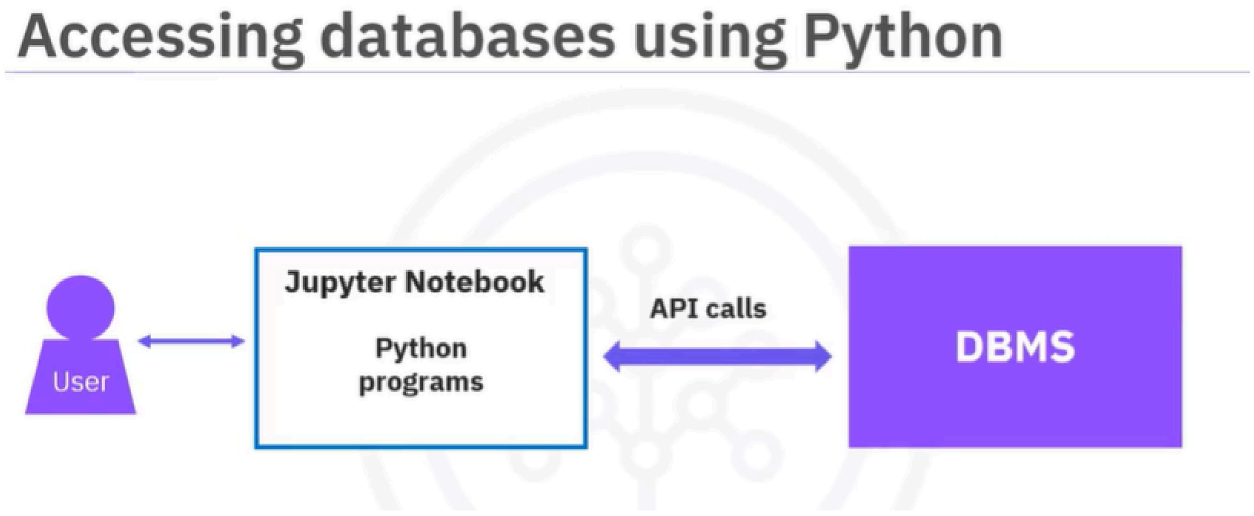
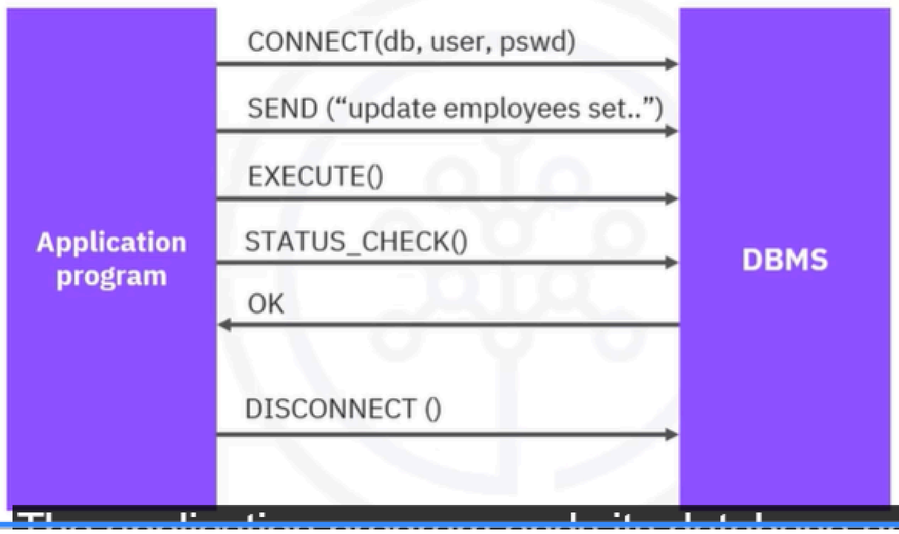
Acess Database using Python

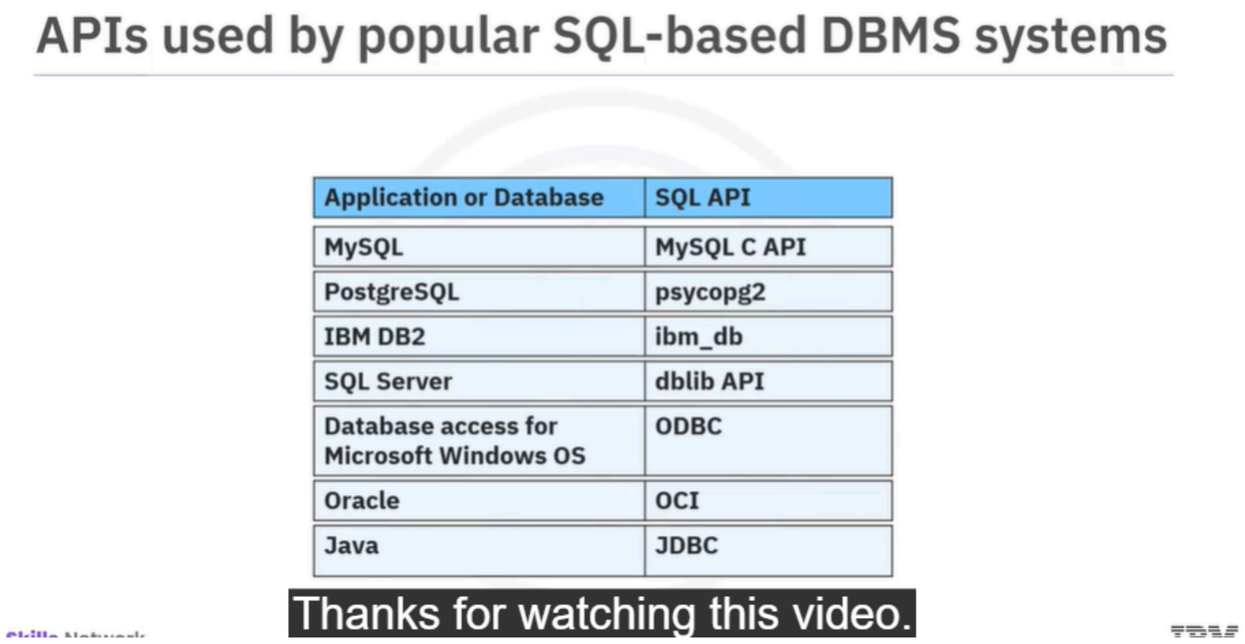


- This is how a typical user accesses databases using Python code written on a Jupyter notebook, a web based editor. There is a mechanism by which the Python program communicates with the DBMS. The Python code connects to the database using API calls. We will explain the basics of SQL APIs and Python DB APIs.

What is a SQL API?



- An application programming interface is a set of functions that you can call to get access to some type of service. The SQL API consists of library function calls as an application programming interface, API, for the DBMS. To pass SQL statements to the DBMS, an application program calls functions in the API, and it calls other functions to retrieve query results and status information from the DBMS. The basic operation of a typical SQL API is illustrated in the figure. The application program begins its database access with one or more API calls that connect the program to the DBMS. To send the SQL statement to the DBMS, the program builds the statement as a text string in a buffer and then makes an API call to pass the buffer contents to the DBMS. The application program makes API calls to check the status of its DBMS request and to handle errors. The application program ends its database access with an API call that disconnects it from the database.



APIs used by popular SQL-based DBMS systems

Application or Database	SQL API
MySQL	MySQL C API
PostgreSQL	psycopg2
IBM DB2	ibm_db
SQL Server	dblib API
Database access for Microsoft Windows OS	ODBC
Oracle	OCI
Java	JDBC

Thanks for watching this video.

- Now, let's learn basic concepts about some of the proprietary APIs used by popular SQL-based DBMS systems. Each database system has its own library. As you can see, the table shows a list of a few applications and corresponding SQL APIs. MySQL C API provides low-level access to the MySQL client server protocol and enables C programs to access database contents. The psycopg2 API connects Python applications in PostgreSQL databases. The IBM_DB API is used to connect Python applications to IBM DB2 databases. The dblib API is used to connect to SQL server databases. ODBC is used for database access for Microsoft Windows OS. OCI is used by Oracle databases. And finally, JDBC is used by Java applications.

How do Python and databases work together?

Here's a simple explanation of how Python and databases work together:

1. **Connecting to a Database**: Python can connect to databases using special tools called APIs (Application Programming Interfaces). These APIs allow

Python to communicate with the database.

2. **Sending Commands**: Once connected, Python can send commands to the database using a language called SQL (Structured Query Language). SQL is used to perform tasks like retrieving data, adding new data, or updating existing data.
3. **Retrieving Data**: After sending a command, Python can receive data back from the database. This data can then be used in Python for analysis or other purposes.
4. **Using Libraries**: Python has libraries like `sqlite3`, `psycopg2`, and `SQLAlchemy` that make it easier to work with databases. These libraries provide functions to connect, send commands, and handle data.
5. **Jupyter Notebooks**: Often, Python code that interacts with databases is written in Jupyter notebooks, which allow you to run code and see results in a user-friendly way.

In summary, Python acts as a bridge to communicate with databases, allowing you to manage and analyze data effectively.

What is the role of APIs in Python database access?

APIs (Application Programming Interfaces) play a crucial role in Python database access by serving as the communication bridge between Python applications and databases. Here's how they work:

1. **Connection**: APIs allow Python to establish a connection to the database. This is the first step in accessing any data stored in the database.
2. **Executing Commands**: Once connected, APIs enable Python to send SQL commands to the database. This includes commands to retrieve, insert, update, or delete data.
3. **Handling Responses**: After executing a command, the API helps Python receive responses from the database. This could be the data requested or confirmation of an action taken.

4. **Error Management**: APIs also provide mechanisms to handle errors that may occur during database operations, ensuring that Python can respond appropriately.
5. **Standardization**: The Python Database API (DB API) provides a standard way for Python programs to interact with different types of databases, making it easier to switch between them without changing much code.

In summary, APIs facilitate the entire process of connecting to, interacting with, and managing databases in Python applications.

What is the function of the Python Database API?

The Python Database API (DB API) serves several important functions for interacting with databases in Python. Here are the key functions:

1. **Standardized Interface**: The DB API provides a consistent interface for different database systems, allowing developers to write code that can work with various databases without significant changes.
2. **Connection Management**: It defines methods for establishing and managing connections to databases, including opening and closing connections.
3. **Executing SQL Commands**: The API allows you to execute SQL commands through methods that send queries to the database and retrieve results.
4. **Handling Transactions**: The DB API supports transaction management, enabling you to commit or roll back changes made during a session, ensuring data integrity.
5. **Error Handling**: It provides a way to handle database errors and exceptions, allowing developers to manage issues that may arise during database operations.
6. **Fetching Data**: The API includes methods for retrieving data from the database, such as fetching rows from query results.

In summary, the Python Database API standardizes how Python interacts with databases, making it easier to connect, execute commands, manage transactions, and handle errors.

How can you handle errors using the Python Database API?

Handling errors using the Python Database API involves using specific techniques to manage exceptions that may occur during database operations. Here's how you can do it:

1. **Try-Except Blocks**: Use `try` and `except` blocks to catch exceptions that may arise when executing database commands. This allows you to handle errors gracefully.

   ```
   try:
       # Code to connect to the database and execute a query
   except SomeDatabaseError as e:
       print(f"An error occurred: {e}")
   ```

2. **Specific Exception Types**: The DB API defines specific exception classes for different types of errors (e.g., `DatabaseError`, `IntegrityError`, `OperationalError`). Catching these specific exceptions helps you understand the nature of the error.

   ```
   from some_database_module import DatabaseError, IntegrityError

   try:
       # Code to execute a query
   except IntegrityError:
       print("Integrity error occurred, possibly a constraint violation.")
   except DatabaseError as e:
       print(f"Database error: {e}")
   ```

3. **Transaction Management**: If an error occurs during a transaction, you can roll back the transaction to maintain data integrity.

```
connection = db.connect()
try:
    cursor = connection.cursor()
    cursor.execute("INSERT INTO table_name VALUES (...)")
    connection.commit()  # Commit if successful
except Exception as e:
    connection.rollback()  # Roll back on error
    print(f"An error occurred: {e}")
finally:
    cursor.close()
    connection.close()
```

4. **Logging Errors**: It’s a good practice to log errors for debugging purposes. You can use Python’s logging module to record error details.

By implementing these techniques, you can effectively handle errors when working with databases in Python, ensuring your application remains robust and user-friendly.