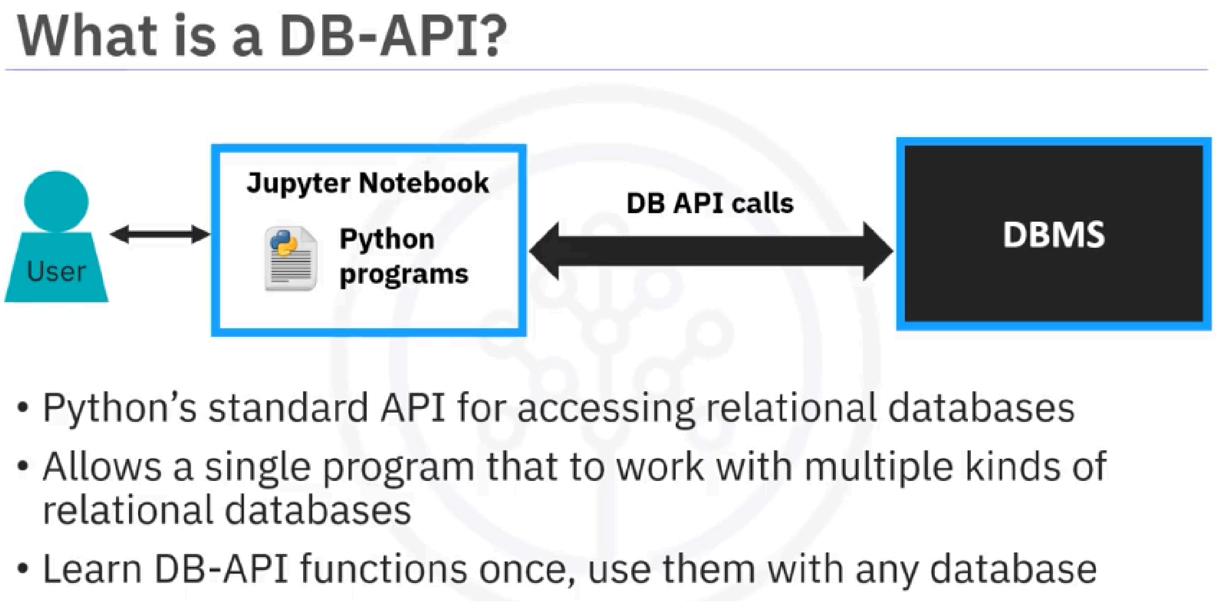
Writing code using DB-API

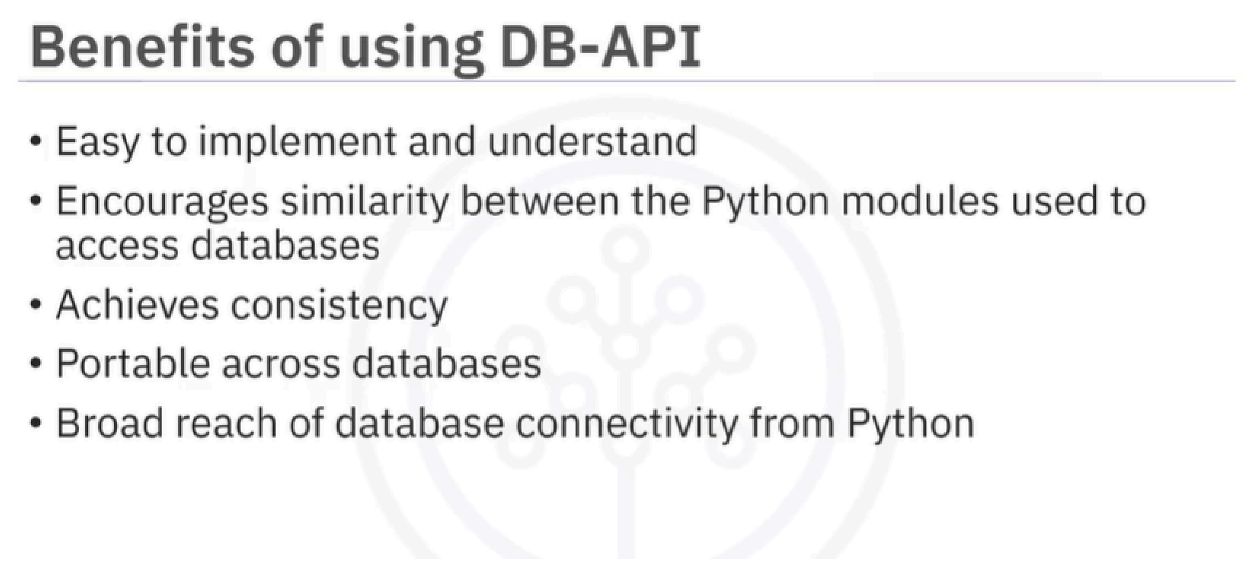


What is a DB-API?

User
Jupyter Notebook
Python programs
DB API calls
DBMS

- Python's standard API for accessing relational databases
- Allows a single program that to work with multiple kinds of relational databases
- Learn DB-API functions once, use them with any database

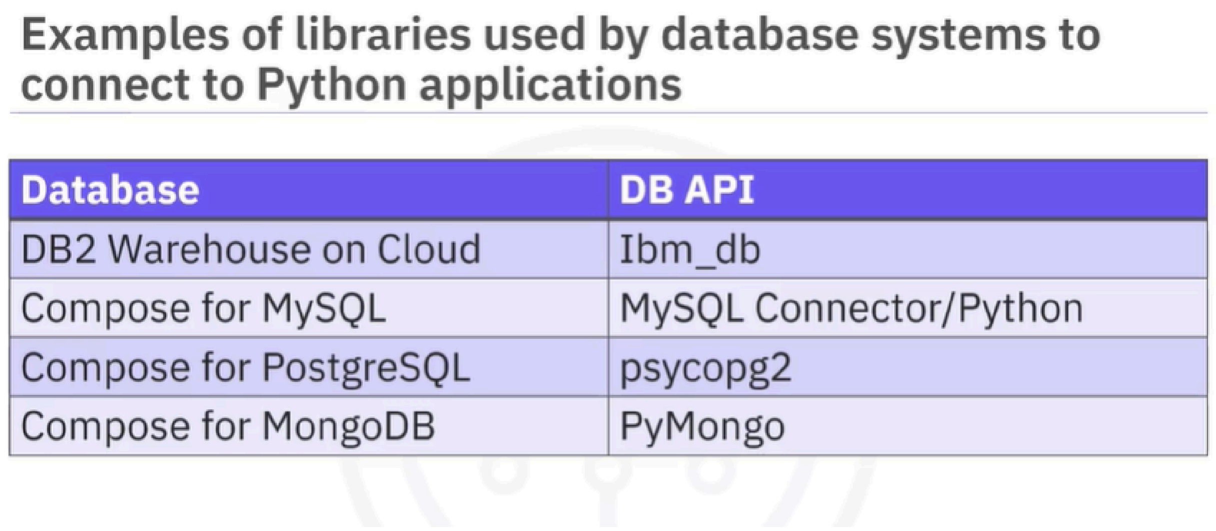
- As we saw in the beginning of this module, the user writes Python programs using a Jupyter Notebook. There is a mechanism by which the Python code communicates with the DBMS. The Python code connects to the database using DB API calls.
- DB API is Python standard API for accessing relational databases. It is a standard that allows you to write a single program that works with multiple kinds of relational databases instead of writing a separate program for each one. If you learn the DB-API functions, then you can apply that knowledge to use any database with Python.



Benefits of using DB-API

- Easy to implement and understand
- Encourages similarity between the Python modules used to access databases
- Achieves consistency
- Portable across databases
- Broad reach of database connectivity from Python

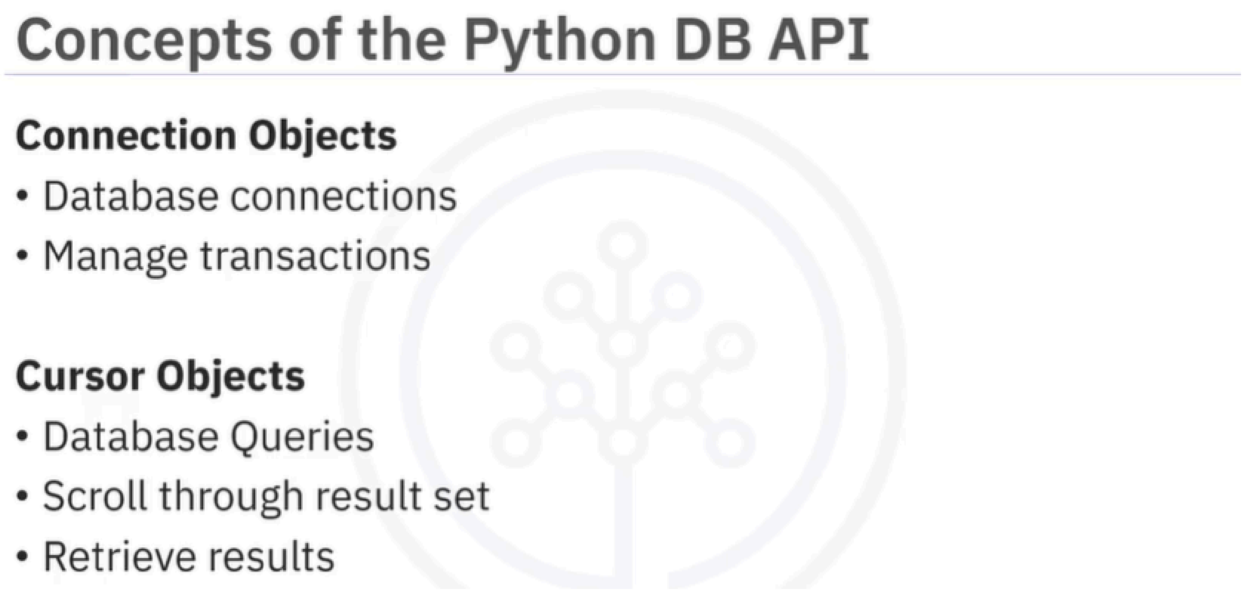
- Here are some advantages of using the DB-API. It's easy to implement and understand. This API has been defined to encourage similarity between the Python modules that are used to access databases. It achieves consistency which leads to more easily understood modules. The code is generally more portable across databases, and it has a broader reach of database connectivity from Python.



Examples of libraries used by database systems to connect to Python applications

Database	DB API
DB2 Warehouse on Cloud	Ibm_db
Compose for MySQL	MySQL Connector/Python
Compose for PostgreSQL	psycopg2
Compose for MongoDB	PyMongo

- As we know, each database system has its own library. As you can see, the table shows a list of a few databases and corresponding DB-APIs to connect to Python applications. The IBM underscore DB library is used to connect to an IBM DB2 database. The mySQL connector Python library is used to connect to a compose from mySQL database. The psychopg2 library is used to connect to a compose from PostgreSQL database. Finally, the PyMongo library is used to connect to a compose from Mongo DB database.



Concepts of the Python DB API

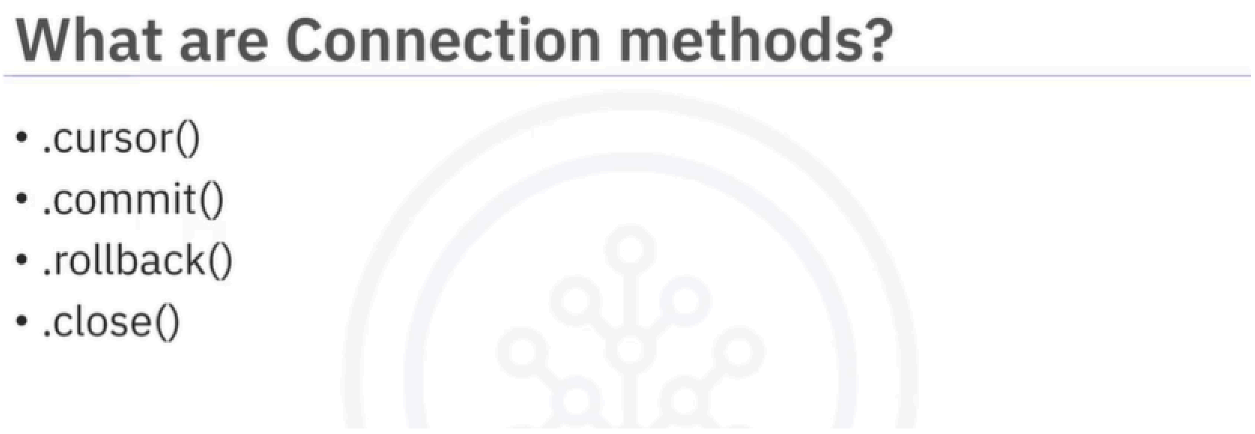
Connection Objects

- Database connections
- Manage transactions

Cursor Objects

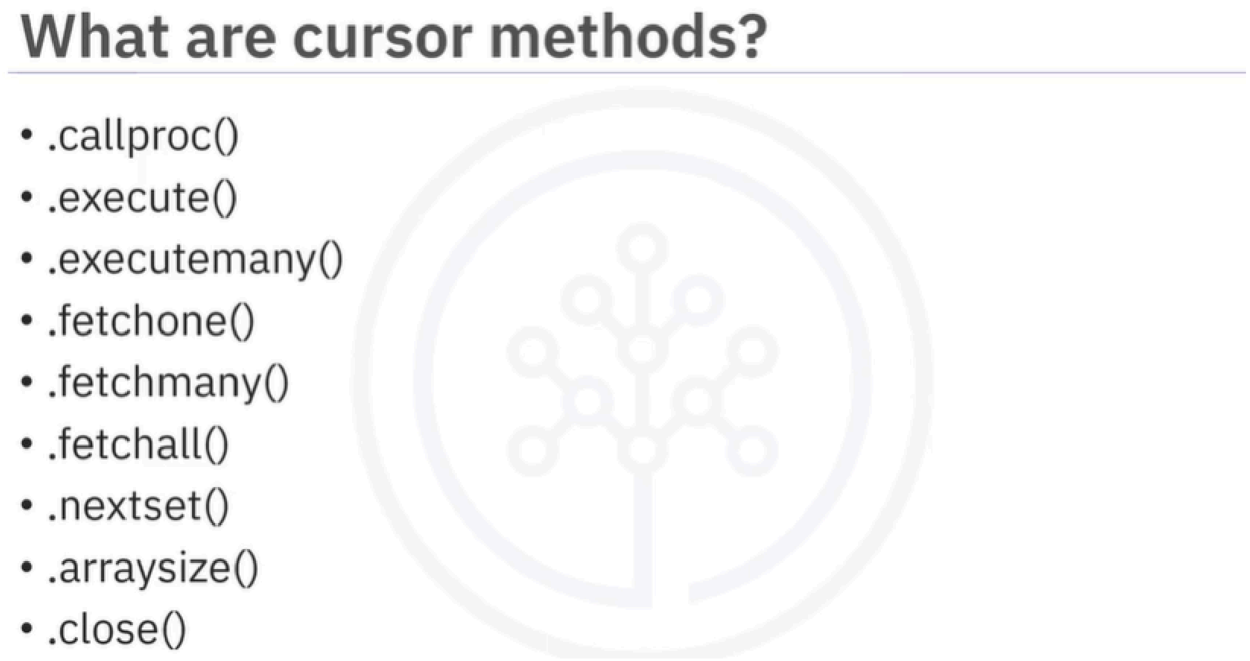
- Database Queries
- Scroll through result set
- Retrieve results

- The two main concepts in the Python DB-API are connection objects and query objects. You use connection objects to connect to a database and manage your transactions. Cursor objects are used to run queries. You open a cursor object and then run queries. The cursor works similar to a cursor in a text processing system, where you scroll down in your result set and get your data into the application. Cursors are used to scan through the results of a database. The DB-API includes a connect constructor for creating a connection to the database. It returns a connection object which is then used by the various connection methods.



What are Connection methods?

- .cursor()
- .commit()
- .rollback()
- .close()

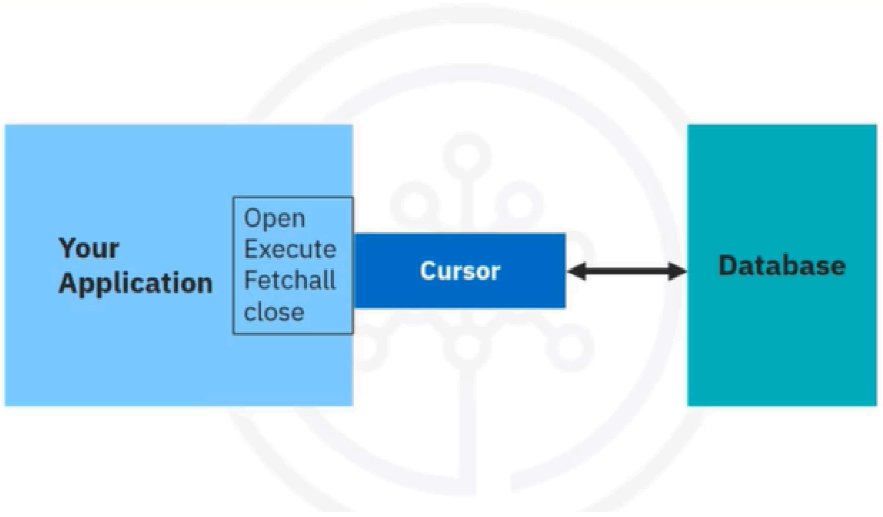


What are cursor methods?

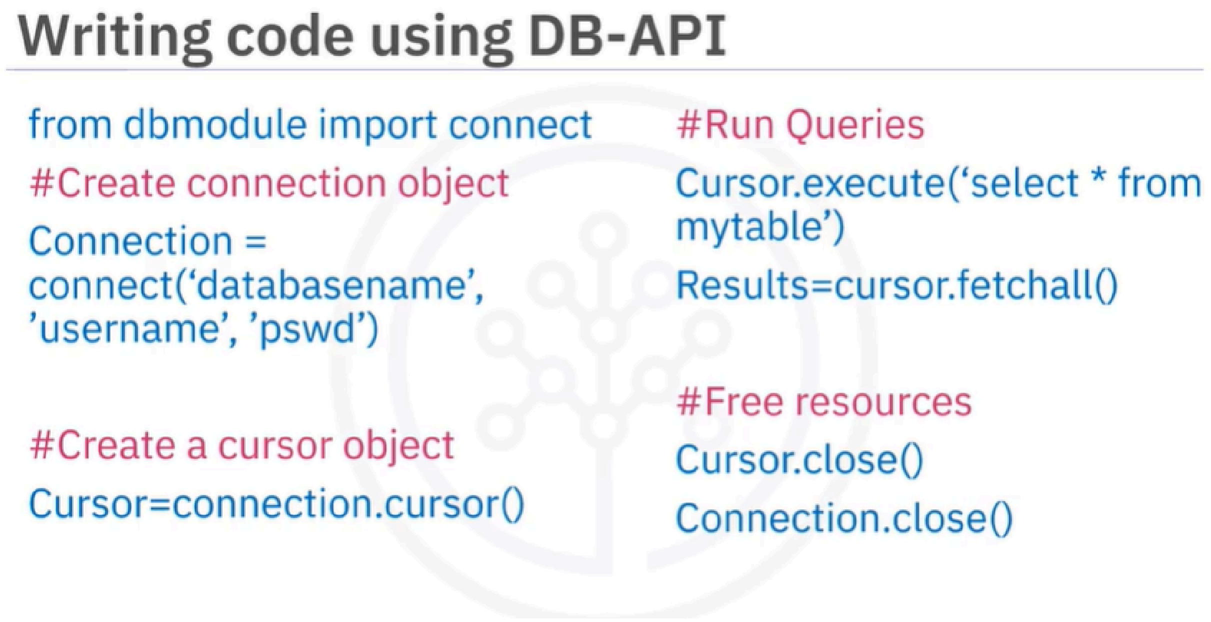
- .callproc()
- .execute()
- .executemany()
- .fetchone()
- .fetchmany()
- .fetchall()
- .nextset()
- .arraysize()
- .close()

- These connection methods are the cursor method, which returns a new cursor object using the connection. The commit method, which is used to commit any pending transaction to the database. The rollback method, which causes the database to roll back to the start of any pending transaction.
- The close method, which is used to close a database connection. These objects represent a database cursor which is used to manage the content of a fetch operation. Cursors created from the same connection are not isolated. That is, any changes done to the database by a cursor are immediately visible by the other cursors. Cursors created from different connections can or cannot be isolated depending on how the transaction support is implemented

What is a database cursor?



- A database cursor is a control structure that enables transversal over the records in a database. It behaves like a file name or file handle in a programming language. Just as a program opens a file to access its contents, it opens a cursor to gain access to the query results. Similarly, the program closes a file to end its access, and closes a cursor to end access to the query results. Another similarity is that just as file handle keeps track of the program's current position within an open file, a cursor keeps track of the program's current position within the query results.



Writing code using DB-API

```
from dbmodule import connect
#Create connection object
Connection =
connect('databasename',
'username', 'pswd')

#Create a cursor object
Cursor=connection.cursor()

#Run Queries
Cursor.execute('select * from
mytable')
Results=cursor.fetchall()

#Free resources
Cursor.close()
Connection.close()
```

- Let's walk through a Python application that uses the DB- API to query a database. First, you import your database module by using the connect API from that module. To open a connection to the database, you use the connect constructor and pass in the parameters, that is database name, username, and password. The connect function returns connection object. After this, you create a cursor object on the connection object. The cursor is used to run queries and fetch results after running the queries using the cursor. We also use the cursor to fetch the results of the query. Finally, when the system is done running the queries, it frees all resources by closing the connection. Remember that it is always important to close connections to avoid unused connections taking up resources.

Let's talk about the concept of DB-API in Python.

- The DB-API is like a universal remote control for different types of databases. Just as a remote can operate various devices like a TV or a DVD player, the DB-API allows you to write Python code that can connect to different databases without needing to learn a new language for each one. This means you can write a single program that works with many databases, making your coding life much easier!

- To illustrate, imagine you have a toolbox with different tools for different tasks. The DB-API is like a multi-tool that combines several functions into one handy device. You can use it to connect to a database, run queries, and manage your data all in one place. For example, you can open a connection to a database, create a cursor to run your queries, and then close the connection when you're done, just like using a tool to complete a project and then putting it away.

What is the purpose of the DB-API in Python?

The purpose of the **DB-API in Python** is to provide a standardized way for Python programs to interact with relational databases. Here are the key points:

- **Consistency**: It creates a uniform interface for different database systems, allowing developers to write code that can work with various databases without needing to change much.
- **Connection Management**: It helps manage connections to databases, enabling you to open, close, and handle transactions effectively.
- **Query Execution**: The DB-API allows you to run SQL queries and retrieve results using cursor objects, making it easier to work with data.
- **Portability**: By using the DB-API, your code becomes more portable across different database systems, which means you can switch databases with minimal changes to your code.

In summary, the DB-API simplifies database interactions in Python, making it easier for developers to work with data across different database platforms.

What would happen if DB-API didn't exist for Python?

If the **DB-API** didn't exist for Python, several challenges would arise:

- **Lack of Standardization**: Each database would require its own unique way of connecting and executing queries. This would lead to a fragmented approach, making it difficult for developers to switch between databases.
- **Increased Complexity**: Developers would need to learn different libraries and APIs for each database system, increasing the learning curve and complexity of database interactions.
- **Code Duplication**: Without a standard interface, developers would likely end up writing similar code for different databases, leading to code duplication and maintenance challenges.
- **Reduced Portability**: Code that works with one database might not work with another, making it harder to share or deploy applications across different environments.
- **Higher Development Time**: The time spent on learning and implementing different database connections would slow down development and increase the likelihood of errors.

In summary, the absence of the DB-API would complicate database interactions in Python, making it less efficient and more challenging for developers.