

SQL Cheat Sheet: Accessing Databases using Python

Topic	Syntax	Description	Example
connect()	<code>sql.connect()</code>	Create a new database and open a database connection to allow sql.py to work with it. Call <code>sql.connect()</code> to create a connection to the database <code>INSTRUCTOR.db</code> in the current working directory, implicitly creating it if it does not exist.	<pre>import sqlite3 con = sqlite3.connect('INSTRUCTOR.db')</pre>
cursor()	<code>con.cursor()</code>	To execute SQL statements and fetch results from SQL queries, use a database cursor. Call <code>con.cursor()</code> to create the Cursor.	<pre>cursor_obj = con.cursor()</pre>
execute()	<code>cursor_obj.execute()</code>	The <code>execute()</code> method in Python SQLite library allows to perform SQL commands, including executing data from a table using a query like "Select * from table_name." When you execute this command, the result is returned as a collection of table data stored in an object, typically in the form of a list of lists.	<pre>cursor_obj.execute("insert into INSTRUCTOR values ('A', 'New', 'Manager', '2020/01', 'USA')")</pre>
fetchall()	<code>cursor_obj.fetchall()</code>	The <code>fetchall()</code> method in Python retrieves all the rows from the result set of a query and presents them as a list of tuples.	<pre>statement = "SELECT * FROM INSTRUCTOR" cursor_obj.execute(statement) result_obj = cursor_obj.fetchall() for row_obj in result_obj: print(row_obj)</pre>
fetchmany()	<code>cursor_obj.fetchmany()</code>	The <code>fetchmany()</code> method retrieves the subsequent group of rows from the result set of a query rather than just a single row. To fetch a few rows from the table, use <code>fetchmany(numberofrows)</code> and mention how many rows you want to fetch.	<pre>statement = "SELECT * FROM INSTRUCTOR" cursor_obj.execute(statement) result_obj = cursor_obj.fetchmany(2) for row_obj in result_obj: print(row_obj)</pre>
read_sql_query()	<code>read_sql_query()</code>	<code>read_sql_query()</code> is a function provided by the Pandas library in Python, and it is not specific to MySQL. It is a generic function used for executing SQL queries on various database systems, including MySQL, and retrieving the results as a Pandas DataFrame.	<pre>df = pd.read_sql_query("select * from instructor", con)</pre>
shape	<code>dataframe.shape</code>	It provides a tuple indicating the shape of a DataFrame or Series, represented as (number of rows, number of columns).	<pre>df.shape</pre>
close()	<code>con.close()</code>	<code>con.close()</code> is a method used to close the connection to a MySQL database. When called, it terminates the connection, releasing any associated resources and ensuring the connection is no longer active. This is important for managing database connections efficiently and preventing resource leaks in your MySQL database interactions.	<pre>con.close()</pre>
CREATE TABLE	<code>CREATE TABLE table_name (column datatype constraints, column datatype constraints, ...);</code>	The <code>CREATE</code> table statement is used to define and create a new table within a database. It specifies the table's name, the structure of its columns (including data types and constraints), and any additional properties such as indexes. This statement essentially sets up the blueprint for organizing and storing data in a structured format within the database.	<pre>CREATE TABLE PERSONALINFORMATION (ID INTEGER PRIMARY KEY, NAME VARCHAR(50), DOB DATE, ADDRESS VARCHAR(100), PHONE VARCHAR(15), EMAIL VARCHAR(50));</pre>
barplot()	<code>seaborn.barplot(x="x-axis-variable", y="y-axis-variable", data=data)</code>	<code>seaborn.barplot()</code> is a function in the Seaborn Python data visualization library used to create a bar plot, also known as a bar chart. It is particularly used to display the relationship between a categorical variable and a numeric variable by showing the average value for each category.	<pre>import seaborn seaborn.barplot(x="Text_Score", y="frequency", data=dataframe)</pre>
read_csv()	<code>df = pd.read_csv('file_path.csv')</code>	<code>read_csv()</code> is a function in Python's Pandas library used for reading data from a Comma-Separated Values (CSV) file and loading it into a Pandas DataFrame. It's a common method for working with tabular data stored in CSV format.	<pre>import pandas df = pandas.read_csv("https://data.cityofchicago.org/resource/3jzq-9hdf.csv")</pre>
to_sql()	<code>df.to_sql('table_name', database)</code>	<code>df.to_sql()</code> is a method in Pandas, a Python data manipulation library used to write the contents of a DataFrame to a SQL database. It allows to take data from a DataFrame and store it externally within a SQL database table.	<pre>import pandas df = pandas.read_csv("https://data.cityofchicago.org/resource/3jzq-9hdf.csv") df.to_sql("Chicago_Information", con = sqlite3.connect('information_schema.db'))</pre>
read_sql()	<code>df = pd.read_sql(sql_query, con)</code>	<code>read_sql()</code> is a function provided by the Pandas library in Python for executing SQL queries and retrieving the results into a DataFrame from an SQL database. It's a convenient way to integrate SQL database interactions into your data analysis workflow.	<pre>sqlalchemy = "select * from INSTRUCTOR" df = pandas.read_sql(statement, con)</pre>

Topic	Syntax	Description	Example
connect()	<code>conn = ibm_db.connect('DSN=IBM011;UID=userid;PWD=password;', '', '')</code>	<code>ibm_db.connect()</code> is a Python function provided by the <code>ibm_db</code> library, which is used for establishing a connection to an IBM DB2 or IBM DB2 Warehouse database. It's commonly used in applications that need to interact with IBM DB2 database from Python.	<pre>import ibm_db conn = ibm_db.connect('DSN=IBM011;UID=userid;PWD=password;', '', '')</pre>
server_info()	<code>ibm_db.server_info()</code>	<code>ibm_db.server_info(conn)</code> is a Python function provided by the <code>ibm_db</code> library, which is used to retrieve information about the IBM DB2 server to which you are connected.	<pre>server = ibm_db.server_info(conn) print ("DBMS_NAME: ", server.DBMS_NAME) print ("TYPE_NAME: ", server.DBMS_NAME) print ("DB_NAME: ", server.DB_NAME)</pre>
close()	<code>con.close()</code>	<code>con.close()</code> is a method used to close the connection to a db2 database. When called, it terminates the connection, releasing any associated resources and ensuring the connection is no longer active. This is important for managing database connections efficiently and preventing resource leaks in your db2 database interactions.	<pre>con.close()</pre>
exec_immediate()	<code>sql_statement = "SQL statement goes here" df = ibm_db.exec_immediate(con, sql_statement)</code>	<code>ibm_db.exec_immediate()</code> is a Python function provided by the <code>ibm_db</code> library, which is used to execute an SQL statement immediately without the need to prepare or bind it. It's commonly used for executing SQL statements that don't require input parameters or don't need to be prepared in advance.	<pre># Using Python drop the table INSTRUCTOR if it exists from a previous attempt. sql_statement = "drop table INSTRUCTOR" df = ibm_db.exec_immediate(con, sql_statement)</pre>