SQL Cheat Sheet: Accessing Databases using Python

SQLite

Topic	Syntax	Description	Example

connect()	`sqlite3.connect()`	Create a new database and open a database connection to allow sqlite3 to work with it. Call `sqlite3.connect()` to create a connection to the database INSTRUCTOR.db in the current working directory, implicitly creating it if it does not exist.	`1. 1`
			`2. 2`
			
			`1. import sqlite3`
			`2. con = sqlite3.connect("INSTRUCTOR.db")`
			
			Copied!

cursor()	`con.cursor()`	To execute SQL statements and fetch results from SQL queries, use a database cursor. Call `con.cursor()` to create the Cursor.	`1. 1`
			
			`1. cursor_obj = con.cursor()`
			
			Copied!

execute()	`cursor_obj.execute()`	The `execute` method in Python's SQLite library allows to perform SQL commands, including retrieving data from a table using a query like "Select * from table_name." When you execute this command, the result is obtained as a collection of table data stored in an object, typically in the form of a list of lists.	`1. 1`
			
			`1. cursor_obj.execute('''insert into INSTRUCTOR values (1, 'Rav', 'Ahuja', 'TORONTO', 'CA')''')`
			
			Copied!

fetchall()	`cursor_obj.fetchall()`	The `fetchall()` method in Python retrieves all the rows from the result set of a query and presents them as a list of tuples.	`1. 1`
			`2. 2`
			`3. 3`
			`4. 4`
			`5. 5`
			
			`1. statement = '''SELECT * FROM INSTRUCTOR'''`
			`2. cursor_obj.execute(statement)`
			`3. output_all = cursor_obj.fetchall()`
			`4. for row_all in output_all:`
			`5.  print(row_all)`
			
			Copied!

fetchmany()	`cursor_obj.fetchmany()`	The `fetchmany()` 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 fetchmany(numberofrows) and mention how many rows you want to fetch.	`1. 1`
			`2. 2`
			`3. 3`
			`4. 4`
			`5. 5`
			
			`1. statement = '''SELECT * FROM INSTRUCTOR'''`
			`2. cursor_obj.execute(statement)`
			`3. output_many = cursor_obj.fetchmany(2)`
			`4. for row_many in output_many:`
			`5.  print(row_many)`
			
			Copied!

read_sql_query()	`read_sql_query()`	`read_sql_query()` 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.	`1. 1`
			
			`1. df = pd.read_sql_query("select * from instructor;", conn)`
			
			Copied!

shape	`dataframe.shape`	It provides a tuple indicating the shape of a DataFrame or Series, represented as (number of rows, number of columns).	`1. 1`
			
			`1. df.shape`
			
			Copied!

close()	`con.close()`	`con.close()` 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.	1. 1 1. con.close() Copied!
CREATE TABLE	`CREATE TABLE table_name ( column1 datatype constraints, column2 datatype constraints, ... );`	The `CREATE 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.	1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 1. CREATE TABLE INTERNATIONAL_STUDENT_TEST_SCORES (2. country VARCHAR(50), 3. first_name VARCHAR(50), 4. last_name VARCHAR(50), 5. test_score INT 6.); Copied!
barplot()	`seaborn.barplot(x="x-axis_variable", y="y-axis_variable", data=data)`	`seaborn.barplot()` 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.	1. 1 2. 2 1. import seaborn 2. seaborn.barplot(x='Test_Score',y='Frequency', data=dataframe) Copied!
read_csv()	`df = pd.read_csv('file_path.csv')`	`read_csv()` 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	1. 1 2. 2 1. import pandas 2. df = pandas.read_csv('https://data.cityofchicago.org/resource/jcxq-k9xf.csv') Copied!
to_sql()	`df.to_sql('table_name', index=False)`	`df.to_sql()` 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 structurally within a SQL database table.	1. 1 2. 2 3. 3 1. import pandas 2. df = pandas.read_csv('https://data.cityofchicago.org/resource/jcxq-k9xf.csv') 3. df.to_sql("chicago_socioeconomic_data", con, if_exists='replace', index=False,method="multi") Copied!
read_sql()	`df = pd.read_sql(sql_query, conn)`	`read_sql()` 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 workflows.	1. 1 2. 2 1. selectQuery = "select * from INSTRUCTOR" 2. df = pandas.read_sql(selectQuery, conn) Copied!

Db2

Topic	Syntax	Description	Example
connect()	`conn = ibm_db.connect('DATABASE=dbname; HOST=hostname;PORT=port;UID=username; PWD=password;', '', '')`	`ibm_db.connect()` is a Python function provided by the ibm_db library, which is used for establishing a connection to an IBM Db2 or IBM Db2 Warehouse database. It's commonly used in applications that	1. 1 2. 2 3. 3 4. 4

		need to interact with IBM Db2 databases from Python.	`1. import ibm_db`
			`2. conn = ibm_db.connect('DATABASE=mydb;`
			`3. HOST=example.com;PORT=50000;UID=myuser;`
			`4. PWD=mypassword;', '', '')`
			Copied!

server_info()	`ibm_db.server_info()`	`ibm_db.server_info(conn)` is a Python function provided by the `ibm_db` library, which is used to retrieve information about the IBM Db2 server to which you are connected.	`1. 1`
			`2. 2`
			`3. 3`
			`4. 4`
			`1. server = ibm_db.server_info(conn)`
			`2. print ("DBMS_NAME: ", server.DBMS_NAME)`
			`3. print ("DBMS_VER:  ", server.DBMS_VER)`
			`4. print ("DB_NAME:   ", server.DB_NAME)`
			Copied!

close()	`con.close()`	`con.close()` 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.	`1. 1`
			`1. con.close()`
			Copied!

exec_immediate()	`sql_statement = "SQL statement goes here"`
	`stmt = ibm_db.exec_immediate(conn, sql_statement)`
		`ibm_db.exec_immediate()` is a Python function provided by the ibm_db 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.	`1. 1`
			`2. 2`
			`3. 3`
			`1. # Lets first drop the table INSTRUCTOR in case it exists from a previous attempt.`
			`2. dropQuery = "drop table INSTRUCTOR"`
			`3. dropStmt = ibm_db.exec_immediate(conn, dropQuery)`
			Copied!

Author(s)

Abhishek Gagneja

D.M Naidu

Changelog

Date	Version	Changed by	Change Description
2023-10-30	1.2	Mary Stenberg	QA Pass with edits
2023-10-16	1.1	Abhishek Gagneja	Updated instruction set
2023-05-08	1.0	D.M.Naidu	Initial Version