**How to Access Databases Using Python**

Databases are powerful tools for data scientists. After completing this module, you'll be able to explain the basic concepts related to using Python to connect to databases. Then you'll create tables, load data and query data using SQL from Jupyter Notebooks, and finally, analyze the data. In the lab assignments, you will learn how to create an instance in the Cloud, connect to a database, query data from the database using SQL, and analyze the data using Python. You will be able to explain the basic concepts related to connecting a Python application to a database. Describe SQL APIs as well as list some of the proprietary APIs used by popular SQL-based DBMS systems. Let's quickly review some of the benefits of using Python, a popular scripting language for connecting to databases. The Python ecosystem is very rich and provides easy to use tools for data science. Some of the most popular packages are NumPy, pandas, matplotlib, and SciPy. Python is easy to learn and has a simple syntax. Due to its open source nature, Python has been ported to many platforms. All your python programs can work on any of these platforms without requiring any changes at all. If you are careful and avoid any system dependent features, Python supports relational database systems. Writing Python code to access databases is made easier by the presence of the Python database API. Commonly referred to as the DB API, and detailed documentation related to Python is easily available. Notebooks are also very popular in the field of data science because they run in an environment that allows creation and sharing of documents that contain live code, equations, visualizations, and explanatory texts. A notebook interface is a virtual notebook environment used for programming. Examples of notebook interfaces include the Mathematica notebook, Maple worksheet, Matlab notebook, IPython Jupyter, R Markdown, Apache Zeppelin, Apache Spark notebook, and the Databricks cloud. In this module, we will be using Jupyter notebooks. The Jupyter notebook is an open source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative texts. Here are some of the advantages of using Jupyter notebooks. Notebook support for over 40 programming languages including Python, R, Julia, and Scala. Notebooks can be shared with others by email, Dropbox, GitHub, and the Jupyter notebook viewer. Your code can produce rich interactive output HTML, images, videos, LaTex, and customized types. You can leverage big data tools such as Apache Spark from Python, R, and Scala, and explore that same data with pandas, scikit-learn, ggplot2, and TensorFlow. 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. 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. Now, lets 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.

**Writing code using DB-API**

After completing this video, you will be able to explain the basic concepts related to the Python DB-API and database cursors. And also write code using DB-APIs. 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's 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. So, if you learn the DB-API functions, then you can apply that knowledge to use any database with 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. 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\_db library is used to connect to an IBM DB2 database.

The MySQL Connector/Python library is used to connect to a Compose for MySQL database. The psycopg2 library is used to connect to a Compose from PostgreSQL database. And finally, the PyMongo library is used to connect to a Compose for MongoDB database. 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. 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.

A database cursor is a control structure that enables traversal 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. 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, the 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.

​

**Connecting to a database using ibm\_db API**

After completing this lesson, you will be able to understand the ibm\_db API, as well as the credentials required to connect to a database using Python. We will also demonstrate how to connect to an IBM DB2 database using Python code written on a Jupyter notebook. The ibm\_db API provides a variety of useful Python functions for accessing and manipulating data in an IBM data server database, including functions for connecting to a database, preparing and issuing SQL statements, fetching rows from result sets, calling stored procedures, committing and rolling back transactions, handling errors and retrieving metadata. The ibm\_db API uses the IBM Data Server Driver for ODBC, and CLI APIs to connect to IBM, DB2, and Informix. We import the ibm\_db library into our Python application. Connecting to the DB2 requires the following information: a driver name, a database name, a host DNS name or IP address, a host port, a connection protocol, a user ID, and a user password. Here is an example of creating a DB2 database connection in Python. We create a connection object DSN, which stores the connection credentials. The connect function of the ibm\_db API will be used to create a non persistent connection. The DSN object is passed as a parameter to the connection function. If a connection has been established with the database, then the code returns connected, as the output otherwise, the output will be unable to connect to database. Then we free all resources by closing the connection. Remember that it is always important to close connections so that we can avoid unused connections taking up resources.

**Creating tables, loading data and querying data**

After completing this lesson, you will be able to understand basic concepts related to creating tables, loading data, and querying data using Python, as well as demonstrate an example of how to perform these tasks using the IBM DB2 on Cloud database and Jupyter notebooks. For this example, we will be using DB2 as the database. We first obtain a connection resource by connecting to the database by using the connect method of the ibm\_db api. There are different ways of creating tables in DB2. One is using the Web console provided by DB2, and the other option is to create tables from any SQL, R, or Python environments. Let's take a look at how to create tables in DB2 from our Python application. Here is a sample table of a commercial Trucks database. Let's see how we can create the Trucks table in the DB2 using Python code. To create a table, we use the ibm\_db.exec\_immediate function. The parameters for the function are connection, which is a valid database connection resource that is returned from the ibm\_db.connect or ibm\_db.pconnect function statement, which is a string that contains the SQL statement, and options which is an optional parameter that includes a dictionary that specifies the type of cursor to return for results sets. Here is the code to create a table called Trucks in Python. We use the ibm\_db.exec\_immediate function of the ibm\_db api. The connection resource that was created is passed as the first parameter to this function. The next parameter is the SQL statement, which is the create table query used to create the Trucks table. The new table created will have five columns, serial\_no will be the primary key. Now let's take a look at loading data. We use the ibm\_db.exec\_immediate function of the ibm\_db api. The connection resource that was created is passed as the first parameter to this function. The next parameter is the SQL statement, which is the insert into query used to insert data in the Trucks table. A new row will be added to the Trucks table. Similarly, we add more rows to the Trucks table using the ibm\_db.exec\_immediate function. Now that your Python code has been connected to a database instance and the database table has been created and populated with data, let's see how we can fetch data from the Trucks table that we created on DB2 using Python code. We use the ibm\_db.exec\_immediate function of the ibm\_db api. The connection resource that was created is passed as the first parameter to this function. The next parameter is the SQL statement, which is the select from table query. The Python code returns the output, which shows the fields of the data in the Trucks table. You can check if the output returned by the select query shown is correct, by referring to the DB2 console. Let's look at how we can use pandas to retrieve data from the database tables. Pandas is a popular Python library that contains high level data structures and manipulation tools designed to make data analysis fast and easy in Python. We load data from the Trucks table into a data frame called DF. A data frame represents a tabular spreadsheet like data structure containing an ordered collection of columns, each of which can be a different value type.

​

**Analyzing Data with Python**

After completing this video, you will be able to understand basic concepts related to performing exploratory analysis on data. We will demonstrate an example of how to store data using the IBM Db2 on Cloud database, and then use Python to do some basic data analysis on this data. In this video, we will be using the McDonald's menu nutritional facts data for popular menu items at McDonald's, while using Python to perform basic exploratory analysis. McDonald's is an American fast food company and the world's largest restaurant chain by revenue. Although McDonald's is known for fast food items such as hamburgers, French fries, soft drinks, milkshakes, and desserts, the company has added to its menu salads, fish, smoothies, and fruit. McDonald's provides nutrition analysis of their menu items to help you balance your McDonald's meal with other foods you eat. The data set used in this lesson has been obtained from the nutritional facts for McDonald's menu from Kaggle. We need to create a table on Db2 to store the McDonald's menu nutrition facts data set that we will be using. We will also be using the console provided by Db2 for this process. There are four steps involved in loading data into a table, source, target, define, and finalize. We first load the spreadsheet into the Db2 using the console. We then select the target schema, and then you will be given an option to load the data into an existing table or create a new table. When you choose to create a new table, you have the option to specify the table name. Next, you will see a preview of the data where you can also define the columns and data types. Review the settings and begin the load. When the loading is complete, you can see the statistics on the loaded data. Next, view the table to explore further. Db2 Warehouse allows you to analyze data using in-database analytics, APIs, RStudio or Python. The data has been loaded into our relational database. You can run Python scripts that retrieve data from and write data to a Db2 database. Such scripts can be powerful tools to help you analyze your data. For example, you can use them to generate statistical models based on data in your database, and to plot the results of these models. In this lesson, we will be using Python scripts that will be run within a Jupyter notebook. Now, after obtaining a connection resource, by connecting to the database, by using the connect method of the IBM\_DB API, we use the SQL select query to verify the number of rows that have been loaded in the table created. The figure shows a snapshot of the output. The output obtained is 260 which is similar to the number of rows in the Db2 console. Now let's see how we can use Pandas to retrieve data from the database tables. We load data from the McDonalds\_nutrition table into the data frame DF using the read\_SQL method. The SQL select query and the connection object are passed as parameters to the read\_SQL method. We can view the first few rows of the data frame DF that we created using the head method. Now it's time to learn about your data. Pandas methods are equipped with a set of common mathematical and statistical methods. Let's use the describe method to view the summary statistics of the data in the data frame, then explore the output of the describe method. We see that there are 260 observations or food items in our data frame. We also see that there are nine unique categories of food items in our data frame. We can also see summary statistics information such as frequency, mean, median, standard deviation, et cetera for the 260 food items across the different variables. For example, the maximum value for total fat is 118. Let's investigate this data further. Let's try to understand one of the nutrients in the food items which is sodium. A main source of sodium is table salt. The average American eats five or more teaspoons of salt each day. This is about 20 times as much as the body needs. Sodium is found naturally in foods, but a lot of it is added during processing and preparation. Many foods that do not taste salty, may still be high in sodium. Large amounts of sodium can be hidden in canned, processed and convenience foods. Sodium controls fluid balance in our bodies, and maintains blood volume and blood pressure. Eating too much sodium may raise blood pressure and cause fluid retention, which could lead to swelling of the legs, and feet, or other health issues. When limiting sodium in your diet, a common target is to eat less than 2,000 milligrams of sodium per day. Now using the nutrition data set for McDonald's, let's do some basic data analysis to answer the question. Which food item has the maximum sodium content? We first use visualization to explore the sodium content of food items. Using the swarm plot method provided by the Seaborne package, we create a categorical scatter plot as shown on the right, then give as the input, category on the x-axis, sodium on the y-axis, and the data will be the data frame DF that contains the nutritional data set from McDonald's. The plot shows the sodium values for the different food items by category. Notice a high value of around 3,600 for sodium on the scatter plot. We will be learning about visualizations later in this module. Let's further explore this high sodium value and identify which food items on the menu have this value for sodium. Let's do some basic data analysis using Python to find which food items on the menu have maximum sodium content. To check the values of sodium levels in the food items within the dataset, we use the code as shown in code 1. The describe method is used to understand the summary statistics associated with sodium. Notice that the maximum value of sodium is given as 3,600. Now let's further explore the row associated with the maximum sodium variable as shown in code 2. We use the idxmax method to compute the index values, at which the maximum value of sodium is obtained in the data frame. We see that the output is 82. Now lets find the item name associated with the 82nd item in our data frame. As shown in code 3, we will use the .at method to find the item name by passing the index of 82 and the column name item, to be returned for the 82nd row. Finally, we find that the food item on the menu that has a highest sodium content is Chicken McNuggets, 40 pieces. Visualizations are very useful for initial data exploration. They can help us understand relationships, patterns, and outliers in the data. Let's first create a scatter plot with protein on the x-axis, and total fat on the y-axis. Scatter plots are very popular visualization tools and show the relationship between two variables with a point for each observation. To do this, we can use the joint plot function provided by the Seaborn package, and give as input, protein on the x-axis and total fat on the y-axis. And the data will be the data frame DF that contains the nutritional data set from McDonald's. The output scatter plot is shown on the right side. The plot has an interesting shape. It shows the correlation between the two variables: protein and fat. Correlation is a measure of association between two variables, and has a value of between -1 and +1. We see that the points on the scatter plot are closer to a straight line in the positive direction. So we have a positive correlation between the two variables. On the top right corner of the scatter plot, we have the values of the Pearson correlation- 0.81 and the significance of the correlation denoted as P - which is a good value that shows the variables are certainly correlated. The plot also shows two histograms: one on the top and the other on the right side. The histogram on the top is that of the variable protein, and the histogram on the right side is that of the variable total fat. We also noticed that there is a point on the scatter plot outside the general pattern. This is a possible outlier. Now let's see how we can visualize data using box plots. Box plots are charts that indicate the distribution of one or more variables. The box in a box plot captures the middle 50 percent of data. Lines and points indicate possible skewness and outliers. Let's create a box plot for sugar. The function we are going to use is box plot from the Seaborn package. We give the column name sugars as input to the box plot function. The output is shown on the right side, where we had the box plot with average values of sugar and food items around 30 grams. We also notice a few outliers that indicate food items with extreme values of sugar. There exist food items in the data set that have sugar content of around 128 grams. Candies maybe among these high sugar content food items on the menu. Now that you know how to do basic exploratory data analysis using Pandas and visualization tools, proceed to the labs in this module where you can practice the concepts learned.