Databases and SQL for Data Science:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Connect SQL (IBM\_DB)**  Create Db2 Service Credentials – Create credentials to access your database instance  Database credentials are required to connect from remote applications like Jupyter notebooks, which are used in the labs and assignment in the last two weeks of the course.  **# Go to your IBM Cloud dashboard:**   * Click on IBM Cloud in the top left corner or go to: <https://console.bluemix.net/dashboard/apps> * If your connection is slow it may take over 30 seconds for the dashboard to fully load. * Locate and click on your Db2 service listed under Cloud Foundry Services. * NOTE: In the example below the service is called “Db2-s6” but your Db2 service may have a different letters/numbers in the suffix e.g. “Db2-xx”   **# Click on Service Credentials in the left menu**  **# Click on the button to create New credentials.**   * In the prompt that comes up click the “Add” button in the bottom right * Check the box to View credentials * Copy and save the credentials making a note of the following:   + port is the database port, db is the database name, host is the hostname of the database instance   + URI is used to connect Jupyter notebooks when using SQL Magic  |  |  | | --- | --- | | db: | "BLUDB" | | host: | "dashdb-txn-sbox-yp-lon02-02.services.eu-gb.bluemix.net" | | hostname: | "dashdb-txn-sbox-yp-lon02-02.services.eu-gb.bluemix.net" | | https\_url: | "https://dashdb-txn-sbox-yp-lon02-02.services.eu-gb.bluemix.net:8443" | | jdbcurl: | "jdbc:db2://dashdb-txn-sbox-yp-lon02-02.services.eu-gb.bluemix.net:50000/BLUDB" | | username: | "tdb58568" "password": "p3w5t-4rm71ldflf""port": 50000 | | ssljdbcurl: | "jdbc:db2://dashdb-txn-sbox-yp-lon02-02.services.eu-gb.bluemix.net:50001/BLUDB:sslConnection=true;" | | uri: | "db2://tdb58568:p3w5t-4rm71ldflf@dashdb-txn-sbox-yp-lon02-02.services.eu-gb.bluemix.net:50000/BLUDB" | | ssldsn: | "DATABASE=BLUDB;HOSTNAME=dashdb-txn-sbox-yp-lon02-02.services.eu-gb.bluemix.net;PORT=50001;PROTOCOL=TCPIP;UID=tdb58568;PWD=p3w5t-4rm71ldflf;Security=SSL;" | | dsn: | "DATABASE=BLUDB;HOSTNAME=dashdb-txn-sbox-yp-lon02-02.services.eu-gb.bluemix.net;PORT=50000;PROTOCOL=TCPIP;UID=tdb58568;PWD=p3w5t-4rm71ldflf;" |   **# Introduction:**  This notebook illustrates how to access a DB2 database on Cloud using Python by following the steps below:   * Import the ibm\_db Python library * Enter the database connection credentials * Create the database connection * Close the database connection   Import the ibm\_db Python library  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. We first import the ibm\_db library into our Python Application.  import ibm\_db # When the command above completes, the ibm\_db library is loaded in your notebook.  # **Identify the database connection credentials, connecting to dashDB or DB2 database requires the following:**  Driver Name, Database name, Host DNS name or IP address, Host port, Connection protocol, User details  **# Replace the placeholder values with your actual Db2 hostname, username, and password:**  dsn\_hostname = "YourDb2Hostname" # e.g.: "dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net"  dsn\_uid = "YourDb2Username" # e.g. "abc12345"  dsn\_pwd = "YoueDb2Password" # e.g. "7dBZ3wWt9XN6$o0J"  ​dsn\_driver = "{IBM DB2 ODBC DRIVER}"  dsn\_database = "BLUDB" # e.g. "BLUDB"  dsn\_port = "50000" # e.g. "50000"  dsn\_protocol = "TCPIP" # i.e. "TCPIP"  **# Create the DB2 database connection**  Ibm\_db API uses the IBM Data Server Driver for ODBC and CLI APIs to connect to IBM DB2 and Informix.  **# Lets build the dsn connection string using the credentials you entered above**  **# DO NOT MODIFY bleow Code. Just RUN it with Shift + Enter**  **# Create the dsn connection string**  dsn = (  "DRIVER={0};"  "DATABASE={1};"  "HOSTNAME={2};"  "PORT={3};"  "PROTOCOL={4};"  "UID={5};"  "PWD={6};").format(dsn\_driver, dsn\_database, dsn\_hostname, dsn\_port, dsn\_protocol, dsn\_uid, dsn\_pwd)  **​# Print the connection string to check correct values are specified**  print(dsn)  **# Now establish the connection to the database**  **# Create database connection**  ​try:  conn = ibm\_db.connect(dsn, "", "")  print ("Connected to database: ", dsn\_database, "as user: ", dsn\_uid, "on host: ", dsn\_hostname)  ​except:  print ("Unable to connect: ", ibm\_db.conn\_errormsg() )  ​**# Retrieve Metadata for the Database Server**  server = ibm\_db.server\_info(conn)​  print ("DBMS\_NAME: ", server.DBMS\_NAME)  print ("DBMS\_VER: ", server.DBMS\_VER)  print ("DB\_NAME: ", server.DB\_NAME)  **#Retrieve Metadata for the Database Client / Driver**  client = ibm\_db.client\_info(conn)  ​print ("DRIVER\_NAME: ", client.DRIVER\_NAME)  print ("DRIVER\_VER: ", client.DRIVER\_VER)  print ("DATA\_SOURCE\_NAME: ", client.DATA\_SOURCE\_NAME)  print ("DRIVER\_ODBC\_VER: ", client.DRIVER\_ODBC\_VER)  print ("ODBC\_VER: ", client.ODBC\_VER)  print ("ODBC\_SQL\_CONFORMANCE: ", client.ODBC\_SQL\_CONFORMANCE)  print ("APPL\_CODEPAGE: ", client.APPL\_CODEPAGE)  print ("CONN\_CODEPAGE: ", client.CONN\_CODEPAGE)  # **Close the Connection**  Remember to close connections so that we can avoid unused connections taking up resources.  ibm\_db.close(conn) |

*Connect SQL (IBM\_DB) through Python Course: Databases and SQL for Data Science - LAB: Create Table*

|  |
| --- |
| Task 4: Create a table in the database  In this step we will create a table in the database with following details:  Image  #Lets first drop the table INSTRUCTOR in case it exists from a previous attempt  dropQuery = "drop table INSTRUCTOR"  #Now execute the drop statement  dropStmt = ibm\_db.exec\_immediate(conn, dropQuery)  Don’t worry if you get this error: If you see an exception/error similar to the following, indicating that INSTRUCTOR is an undefined name, that's okay. It just implies that the INSTRUCTOR table does not exist in the table - which would be the case if you had not created it previously.  Exception: [IBM][CLI Driver][DB2/LINUXX8664] SQL0204N "ABC12345.INSTRUCTOR" is an undefined name. SQLSTATE=42704 SQLCODE=-204  #Construct the Create Table DDL statement - replace the ... with rest of the statement  createQuery = "create table INSTRUCTOR(id INTEGER PRIMARY KEY NOT NULL, fname ...)"  #Now fill in the name of the method and execute the statement  createStmt = ibm\_db.replace\_with\_name\_of\_execution\_method(conn, createQuery)  Task 5: Insert data into the table  In this step we will insert some rows of data into the table. The INSTRUCTOR table we created in the previous step contains 3 rows of data:    We will start by inserting just the first row of data, i.e. for instructor Rav Ahuja  #Construct the query  insertQuery = "insert into INSTRUCTOR values (1, 'Rav', 'Ahuja', 'TORONTO', 'CA'), (2, 'Raul', 'Chong', 'Markham', 'CA'),(3, 'Hima', 'Vasudeven', 'Chicago', 'CA')"  #execute the insert statement  insertStmt = ibm\_db.exec\_immediate(conn, insertQuery)  #Construct the query that retrieves all rows from the INSTRUCTOR table  selectQuery = "select \* from INSTRUCTOR"  #Execute the statement  selectStmt = ibm\_db.exec\_immediate(conn, selectQuery)  #Fetch the Dictionary (for the first row only)  ibm\_db.fetch\_both(selectStmt)  #Fetch the rest of the rows and print the ID and FNAME for those rows  while ibm\_db.fetch\_row(selectStmt) != False:  print (" ID:", ibm\_db.result(selectStmt, 0), " FNAME:", ibm\_db.result(selectStmt, "FNAME"))  #Construct the query that update  updateQuery = "UPDATE INSTRUCTOR SET CITY = 'MOOSETOWN' where ID = 1"  #Execute the statement  updateStmt = ibm\_db.exec\_immediate(conn, updateQuery)  updateQuery = "update INSTRUCTOR set CITY='MOOSETOWN' where FNAME='Rav'"  updateStmt = ibm\_db.exec\_immediate(conn, updateQuery))  Task 7: Retrieve data into Pandas  In this step we will retrieve the contents of the INSTRUCTOR table into a Pandas data frame  #connection for pandas  pconn = ibm\_db\_dbi.Connection(conn)  #query statement to retrieve all rows in INSTRUCTOR table  selectQuery = "select \* from INSTRUCTOR"  #retrieve the query results into a pandas dataframe  pdf = pandas.read\_sql(selectQuery, pconn)  #print just the LNAME for first row in the pandas data frame  pdf.LNAME[0]  #print the entire data frame  pdf or pdf.Head()  Once the data is in a Pandas dataframe, you can do the typical pandas operations on it. For example you can use the shape method to see how many rows and columns are in the dataframe  pdf.shape  pdf.describe()  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.  ibm\_db.close(conn) |
| Accessing Databases with SQL Magic  After using this notebook, you will know how to perform simplified database access using SQL "magic". You will connect to a Db2 database, issue SQL commands to create tables, insert data, and run queries, as well as retrieve results in a Python dataframe.  To communicate with SQL Databases from within a JupyterLab notebook, we can use the SQL "magic" provided by the ipython-sql extension. "Magic" is JupyterLab's term for special commands that start with "%". Below, we'll use the load\_\_\_ext magic to load the ipython-sql extension. In the lab environemnt provided in the course the ipython-sql extension is already installed and so is the ibm\_db\_sa driver.  %load\_ext sql  Now we have access to SQL magic. With our first SQL magic command, we'll connect to a Db2 database. However, in order to do that, you'll first need to retrieve or create your credentials to access your Db2 database.  Image  This image shows the location of your connection string if you're using Db2 on IBM Cloud. If you're using another host the format is: username:password@hostname:port/database-name  # Enter your Db2 credentials in the connection string below  # Recall you created Service Credentials in Part III of the first lab of the course in Week 1  # i.e. from the uri field in the Service Credentials copy everything after db2:// (but remove the double quote at the end)  # for example, if your credentials are as in the screenshot above, you would write:  # %sql ibm\_db\_sa://my-username:my-password@dashdb-txn-sbox-yp-dal09-03.services.dal.bluemix.net:50000/BLUDB  # Note the ibm\_db\_sa:// prefix instead of db2://  # This is because JupyterLab's ipython-sql extension uses sqlalchemy (a python SQL toolkit)  # which in turn uses IBM's sqlalchemy dialect: ibm\_db\_sa  %sql ibm\_db\_sa://tdb58568:p3w5t-4rm71ldflf@dashdb-txn-sbox-yp-lon02-02.services.eu-gb.bluemix.net:50000/BLUDB  For convenience, we can use %%sql (two %'s instead of one) at the top of a cell to indicate we want the entire cell to be treated as SQL. Let's use this to create a table and fill it with some test data for experimenting. Sample query full attached  %%sql  ​  CREATE TABLE INTERNATIONAL\_STUDENT\_TEST\_SCORES (  country VARCHAR(50),  first\_name VARCHAR(50),  last\_name VARCHAR(50),  test\_score INT  );  INSERT INTO INTERNATIONAL\_STUDENT\_TEST\_SCORES (country, first\_name, last\_name, test\_score)  VALUES  ('United States', 'Marshall', 'Bernadot', 54),  ('Ghana', 'Celinda', 'Malkin', 51),  ('Ukraine', 'Guillermo', 'Furze', 53)  Using Python Variables in your SQL Statements  You can use python variables in your SQL statements by adding a ":" prefix to your python variable names.  For example, if I have a python variable country with a value of "Canada", I can use this variable in a SQL query to find all the rows of students from Canada.  country = "Canada"  %sql select \* from INTERNATIONAL\_STUDENT\_TEST\_SCORES where country = :country  Assigning the Results of Queries to Python Variables  You can use the normal python assignment syntax to assign the results of your queries to python variables.  For example, I have a SQL query to retrieve the distribution of test scores (i.e. how many students got each score). I can assign the result of this query to the variable test\_score\_distribution using the = operator.  test\_score\_distribution = %sql SELECT test\_score as "Test Score", count(\*) as "Frequency" from INTERNATIONAL\_STUDENT\_TEST\_SCORES GROUP BY test\_score;  test\_score\_distribution  Converting Query Results to DataFrames  You can easily convert a SQL query result to a pandas dataframe using the DataFrame() method. Dataframe objects are much more versatile than SQL query result objects. For example, we can easily graph our test score distribution after converting to a dataframe.  dataframe = test\_score\_distribution.DataFrame()  ​  %matplotlib inline  # uncomment the following line if you get an module error saying seaborn not found  # !pip install seaborn  import seaborn  ​  plot = seaborn.barplot(x='Test Score',y='Frequency', data=dataframe)  Now you know how to work with Db2 from within JupyterLab notebooks using SQL "magic"!  %%sql  ​  -- Feel free to experiment with the data set provided in this notebook for practice:  SELECT country, first\_name, last\_name, test\_score FROM INTERNATIONAL\_STUDENT\_TEST\_SCORES; |
| Lab: Analyzing a real world data-set with SQL and Python  Introduction  This notebook shows how to store a dataset into a database using and analyse data using SQL and Python. In this lab you will:  Understand a dataset of selected socioeconomic indicators in Chicago  Learn how to store data in an Db2 database on IBM Cloud instance  Solve example problems to practice your SQL skills  Selected Socioeconomic Indicators in Chicago  The city of Chicago released a dataset of socioeconomic data to the Chicago City Portal. This dataset contains a selection of six socioeconomic indicators of public health significance and a “hardship index,” for each Chicago community area, for the years 2008 – 2012.  Scores on the hardship index can range from 1 to 100, with a higher index number representing a greater level of hardship.  A detailed description of the dataset can be found on the city of Chicago's website, but to summarize, the dataset has the following variables:  Community Area Number (ca): Used to uniquely identify each row of the dataset  Community Area Name (community\_area\_name): The name of the region in the city of Chicago  Percent of Housing Crowded (percent\_of\_housing\_crowded): Percent of occupied housing units with more than one person per room  Percent Households Below Poverty (percent\_households\_below\_poverty): Percent of households living below the federal poverty line  Percent Aged 16+ Unemployed (percent\_aged\_16\_unemployed): Percent of persons over the age of 16 years that are unemployed  Percent Aged 25+ without High School Diploma (percent\_aged\_25\_without\_high\_school\_diploma): Percent of persons over the age of 25 years without a high school education  Percent Aged Under 18 or Over 64:Percent of population under 18 or over 64 years of age (percent\_aged\_under\_18\_or\_over\_64): (ie. dependents)  Per Capita Income (per\_capita\_income\_): Community Area per capita income is estimated as the sum of tract-level aggragate incomes divided by the total population  Hardship Index (hardship\_index): Score that incorporates each of the six selected socioeconomic indicators  In this Lab, we'll take a look at the variables in the socioeconomic indicators dataset and do some basic analysis with Python.  Connect to the database  Let us first load the SQL extension and establish a connection with the database  %load\_ext sql  # Remember the connection string is of the format:  # %sql ibm\_db\_sa://my-username:my-password@my-hostname:my-port/my-db-name  # Enter the connection string for your Db2 on Cloud database instance below  # i.e. copy after db2:// from the URI string in Service Credentials of your Db2 instance. Remove the double quotes at the end.  %sql ibm\_db\_sa://tdb58568:p3w5t-4rm71ldflf@dashdb-txn-sbox-yp-lon02-02.services.eu-gb.bluemix.net:50000/BLUDB  Store the dataset in a Table  In many cases the dataset to be analyzed is available as a .CSV (comma separated values) file, perhaps on the internet. To analyze the data using SQL, it first needs to be stored in the database.  We will first read the dataset source .CSV from the internet into pandas dataframe  Then we need to create a table in our Db2 database to store the dataset.  The PERSIST command in SQL "magic" simplifies the process of table creation and writing the data from a pandas dataframe into the table  import pandas  chicago\_socioeconomic\_data = pandas.read\_csv('https://data.cityofchicago.org/resource/jcxq-k9xf.csv')  %sql PERSIST chicago\_socioeconomic\_data  You can verify that the table creation was successful by making a basic query like:  %sql SELECT \* FROM chicago\_socioeconomic\_data limit 5;  How many rows are in the dataset?  %sql SELECT count(\*) FROM chicago\_socioeconomic\_data;  How many community areas in Chicago have a hardship index greater than 50.0?  %sql SELECT COUNT(\*) FROM chicago\_socioeconomic\_data where hardship\_index > 50.0 ;  What is the maximum value of hardship index in this dataset?  %sql SELECT MAX(hardship\_index) FROM chicago\_socioeconomic\_data;  ​  ​Which community area which has the highest hardship index?  %%sql SELECT community\_area\_name FROM chicago\_socioeconomic\_data  where hardship\_index = (SELECT MAX(hardship\_index) FROM chicago\_socioeconomic\_data);  ​## or another options:  %sql SELECT community\_area\_name FROM chicago\_socioeconomic\_data where hardship\_index=98.0  ​  %sql SELECT community\_area\_name FROM chicago\_socioeconomic\_data ORDER BY hardship\_index DESC NULLS LAST FETCH FIRST ROW ONLY;  ​  Which Chicago community areas have per-capita incomes greater than $60,000?  %sql select community\_area\_name from chicago\_socioeconomic\_data where per\_capita\_income\_ > 60000  ​  Create a scatter plot using the variables per\_capita\_income\_ and hardship\_index. Explain the correlation between the two variables.  ​  # if the import command gives ModuleNotFoundError: No module named 'seaborn'  # then uncomment the following line i.e. delete the # to install the seaborn package  # !pip install seaborn  import matplotlib.pyplot as plt  %matplotlib inline  import seaborn as sns  ​  income\_vs\_hardship = %sql SELECT per\_capita\_income\_, hardship\_index FROM chicago\_socioeconomic\_data;  plot = sns.jointplot(x='per\_capita\_income\_',y='hardship\_index', data=income\_vs\_hardship.DataFrame())  ​  Correct answer: You can see that as Per Capita Income rises as the Hardship Index decreases. We see that the points on the scatter plot are somewhat closer to a straight line in the negative direction, so we have a negative correlation between the two variables.  -->  ​  Conclusion  Now that you know how to do basic exploratory data analysis using SQL and python visualization tools, you can further explore this dataset to see how the variable per\_capita\_income\_ is related to percent\_households\_below\_poverty and percent\_aged\_16\_unemployed. Try to create interesting visualizations!  #!pip install seaborn  import matplotlib.pyplot as plt  %matplotlib inline  import seaborn as sns  income\_vs\_poverty = %sql SELECT per\_capita\_income\_, percent\_households\_below\_poverty FROM chicago\_socioeconomic\_data;  plot = sns.jointplot(x='per\_capita\_income\_',y='percent\_households\_below\_poverty', data=income\_vs\_poverty.DataFrame())  import matplotlib.pyplot as plt  %matplotlib inline  import seaborn as sns  plot = sns.set\_style("whitegrid")  ax = sns.boxplot(x = chicago\_socioeconomic\_data["per\_capita\_income\_"])  Scatter plot by Category  import matplotlib.pyplot as plt  %matplotlib inline  import seaborn as sns  community\_vs\_poverty = %sql SELECT community\_area\_name, percent\_households\_below\_poverty FROM chicago\_socioeconomic\_data where percent\_households\_below\_poverty > 30.0 ;  plot = sns.swarmplot(x='community\_area\_name',y='percent\_households\_below\_poverty', data=community\_vs\_poverty.DataFrame())  plt.setp(plot.get\_xticklabels(),rotation=70)  plt.title('Dpatel')  plt.show()  chicago\_socioeconomic\_data['percent\_households\_below\_poverty'].describe()  chicago\_socioeconomic\_data['percent\_households\_below\_poverty'].idxmax()  chicago\_socioeconomic\_data.at[53,'community\_area\_name']  chicago\_socioeconomic\_data.iloc[53,0:3]  chicago\_socioeconomic\_data.iloc[50:54,0:3]  chicago\_socioeconomic\_data.loc[50:54,'ca':'community\_area\_name']  Summary  In this lab you learned how to store a real-world data set from the internet in a database (Db2 on IBM Cloud), gain insights into data using SQL queries. You also visualized a portion of the data in the database to see what story it tells. |