Access DB2 on Cloud using Python

Estimated time needed: **15** minutes

Objectives

After completing this lab you will be able to:

- Create a table
- Insert data into the table
- Query data from the table
- Retrieve the result set into a pandas dataframe
- Close the database connection

Notice: Please follow the instructions given in the first Lab of this course to Create a database service instance of Db2 on Cloud.

Task 1: 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 import the ibm_db library into our Python Application

The following required modules are pre-installed in the Skills Network Labs environment. However if you run this notebook commands in a different Jupyter environment (e.g. Watson Studio or Ananconda) you may need to install these libraries by removing the `#` sign before `!pip` in the code cell below.

In [1]:
```
# These libraries are pre-installed in SN Labs. If running in another environment p
# !pip install --force-reinstall ibm_db==3.1.0 ibm_db_sa==0.3.3
# Ensure we don't load_ext with sqlalchemy>=1.4 (incompadible)
```

```
# !pip uninstall sqlalchemy==1.4 -y && pip install sqlalchemy==1.3.24
# !pip install ipython-sql
```

In [2]:
```
!pip install --force-reinstall ibm_db ibm_db_sa
import ibm_db
```

```
Collecting ibm_db
  Using cached ibm_db-3.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.w
hl.metadata (1.4 kB)
Collecting ibm_db_sa
  Using cached ibm_db_sa-0.4.1-py3-none-any.whl.metadata (5.3 kB)
Collecting sqlalchemy>=0.7.3 (from ibm_db_sa)
  Using cached SQLAlchemy-2.0.36-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86
_64.whl.metadata (9.7 kB)
Collecting typing-extensions>=4.6.0 (from sqlalchemy>=0.7.3->ibm_db_sa)
  Using cached typing_extensions-4.12.2-py3-none-any.whl.metadata (3.0 kB)
Collecting greenlet!=0.4.17 (from sqlalchemy>=0.7.3->ibm_db_sa)
  Using cached greenlet-3.1.1-cp311-cp311-manylinux_2_24_x86_64.manylinux_2_28_x86_6
4.whl.metadata (3.8 kB)
Using cached ibm_db-3.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(43.6 MB)
Using cached ibm_db_sa-0.4.1-py3-none-any.whl (31 kB)
Using cached SQLAlchemy-2.0.36-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl (3.2 MB)
Using cached greenlet-3.1.1-cp311-cp311-manylinux_2_24_x86_64.manylinux_2_28_x86_64.
whl (602 kB)
Using cached typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Installing collected packages: ibm_db, typing-extensions, greenlet, sqlalchemy, ibm_
db_sa
  Attempting uninstall: ibm_db
    Found existing installation: ibm_db 3.2.3
    Uninstalling ibm_db-3.2.3:
      Successfully uninstalled ibm_db-3.2.3
  Attempting uninstall: typing-extensions
    Found existing installation: typing_extensions 4.12.2
    Uninstalling typing_extensions-4.12.2:
      Successfully uninstalled typing_extensions-4.12.2
  Attempting uninstall: greenlet
    Found existing installation: greenlet 3.1.1
    Uninstalling greenlet-3.1.1:
      Successfully uninstalled greenlet-3.1.1
  Attempting uninstall: sqlalchemy
    Found existing installation: SQLAlchemy 2.0.36
    Uninstalling SQLAlchemy-2.0.36:
      Successfully uninstalled SQLAlchemy-2.0.36
  Attempting uninstall: ibm_db_sa
    Found existing installation: ibm_db_sa 0.4.1
    Uninstalling ibm_db_sa-0.4.1:
      Successfully uninstalled ibm_db_sa-0.4.1
Successfully installed greenlet-3.1.1 ibm_db-3.2.3 ibm_db_sa-0.4.1 sqlalchemy-2.0.36
typing-extensions-4.12.2
```

When the command above completes, the `ibm_db` library is loaded in your notebook.

Task 2: Identify the database connection credentials

Connecting to dashDB or DB2 database requires the following information:

- Driver Name
- Database name
- Host DNS name or IP address
- Host port
- Connection protocol
- User ID
- User Password

Notice: To obtain credentials please refer to the instructions given in the first Lab of this course

Now enter your database credentials below

Replace the placeholder values in angular brackets <> below with your actual database credentials

e.g. replace "database" with "BLUDB"

In [3]:
```
#Replace the placeholder values with your actual Db2 hostname, username, and passwo
dsn_hostname = "1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n41cmd0nqnrk39u98g.databases
dsn_uid = "jwg19774"        # e.g. "abc12345"
dsn_pwd = "RsgH6FXWZ3OsrUD9"      # e.g. "7dBZ3wWt9XN6$o0J"

dsn_driver = "{IBM DB2 ODBC DRIVER}"
dsn_database = "bludb"            # e.g. "BLUDB"
dsn_port = "32286"                # e.g. "32733"
dsn_protocol = "TCPIP"            # i.e. "TCPIP"
dsn_security = "SSL"              #i.e. "SSL"
```

Task 3: Create the database connection

Ibm_db API uses the IBM Data Server Driver for ODBC and CLI APIs to connect to IBM DB2 and Informix.

Create the database connection

In [4]:
```
#Create database connection
#DO NOT MODIFY THIS CELL. Just RUN it with Shift + Enter
dsn = (
    "DRIVER={0};"
    "DATABASE={1};"
    "HOSTNAME={2};"
    "PORT={3};"
    "PROTOCOL={4};"
    "UID={5};"
    "PWD={6};"
    "SECURITY={7};").format(dsn_driver, dsn_database, dsn_hostname, dsn_port, dsn_p
```

```
try:
    conn = ibm_db.connect(dsn, "", "")
    print ("Connected to database: ", dsn_database, "as user: ", dsn_uid, "on host:

except:
    print ("Unable to connect: ", ibm_db.conn_errormsg() )
```

```
Connected to database:  bludb as user:  jwg19774 on host:  1bbf73c5-d84a-4bb0-85b9-a
b1a4348f4a4.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud
```

Task 4: Create a table in the database

In this step we will create a table in the database with following details:

Table definition

INSTRUCTOR

COLUMN NAME	DATA TYPE	NULLABLE
ID	INTEGER	N
FNAME	VARCHAR	Y
LNAME	VARCHAR	Y
CITY	VARCHAR	Y
CCODE	CHARACTER	Y

In [5]:
```
#Lets first drop the table INSTRUCTOR in case it exists from a previous attempt
dropQuery = "drop table INSTRUCTOR"

#Now execute the drop statment
dropStmt = ibm_db.exec_immediate(conn, dropQuery)
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
Cell In[5], line 5
      2 dropQuery = "drop table INSTRUCTOR"
      4 #Now execute the drop statment
----> 5 dropStmt = ibm_db.exec_immediate(conn, dropQuery)

Exception: [IBM][CLI Driver][DB2/LINUXX8664] SQL0204N  "JWG19774.INSTRUCTOR" is an u
ndefined name.  SQLSTATE=42704 SQLCODE=-204
```

Dont 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

In [7]:
```
#Construct the Create Table DDL statement - replace the ... with rest of the statem
createQuery = "create table INSTRUCTOR(ID INTEGER PRIMARY KEY NOT NULL, FNAME VARCH


#Now fill in the name of the method and execute the statement
createStmt = ibm_db.exec_immediate(conn,createQuery)
```

▶ Click here for the solution

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:

INSTRUCTOR

ID INTEGER	FNAME VARCHAR(20)	LNAME VARCHAR(20)	CITY VARCHAR(20)	CCODE CHARACTER(2)
1	Rav	Ahuja	TORONTO	CA
2	Raul	Chong	Markham	CA
3	Hima	Vasudevan	Chicago	US

We will start by inserting just the first row of data, i.e. for instructor Rav Ahuja

In [8]:
```
#Construct the query - replace ... with the insert statement
insertQuery = "insert into INSTRUCTOR values (1, 'Rav', 'Ahuja', 'TORONTO', 'CA')"

#execute the insert statement
insertStmt = ibm_db.exec_immediate(conn, insertQuery)
```

▶ Click here for the solution

Now use a single query to insert the remaining two rows of data

In [9]:
```
#replace ... with the insert statement that inerts the remaining two rows of data
insertQuery2 = "insert into INSTRUCTOR values (2, 'Raul', 'Chong', 'Markham', 'CA')

#execute the statement
insertStmt2 = ibm_db.exec_immediate(conn, insertQuery2)
```

▶ Click here for the solution

Task 6: Query data in the table

In this step we will retrieve data we inserted into the INSTRUCTOR table.

In [10]:
```
#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) - replace ... with your code
ibm_db.fetch_both(selectStmt)
```

Out[10]:
```
{'ID': 1,
 0: 1,
 'FNAME': 'Rav',
 1: 'Rav',
 'LNAME': 'Ahuja',
 2: 'Ahuja',
 'CITY': 'TORONTO',
 3: 'TORONTO',
 'CCODE': 'CA',
 4: 'CA'}
```

▶ Click here for the solution

In [11]:
```
#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(selectSt
```

```
ID: 2  FNAME: Raul
ID: 3  FNAME: Hima
```

▶ Click here for the solution

Bonus: now write and execute an update statement that changes the Rav's CITY to MOOSETOWN

In [12]:
```
#Enter your code below
updateQuery = "update INSTRUCTOR set CITY='MOOSETOWN' where FNAME='Rav'"
updateStmt = ibm_db.exec_immediate(conn, updateQuery)
```

▶ Click here for the solution

Task 7: Retrieve data into Pandas

In this step we will retrieve the contents of the INSTRUCTOR table into a Pandas dataframe

In [13]:
```
!pip install pandas
import pandas
import ibm_db_dbi
```

```
Collecting pandas
  Downloading pandas-2.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.wh
l.metadata (89 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 89.9/89.9 kB 8.7 MB/s eta 0:00:00
Collecting numpy>=1.23.2 (from pandas)
  Downloading numpy-2.1.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.wh
l.metadata (60 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.9/60.9 kB 7.1 MB/s eta 0:00:00
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/conda/lib/python3.11/s
ite-packages (from pandas) (2.9.0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.11/site-packag
es (from pandas) (2024.1)
Collecting tzdata>=2022.7 (from pandas)
  Downloading tzdata-2024.2-py2.py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.11/site-packages
(from python-dateutil>=2.8.2->pandas) (1.16.0)
Downloading pandas-2.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(13.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.1/13.1 MB 122.2 MB/s eta 0:00:0000:01
00:01
Downloading numpy-2.1.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(16.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.3/16.3 MB 113.0 MB/s eta 0:00:0000:01
00:01
Downloading tzdata-2024.2-py2.py3-none-any.whl (346 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 346.6/346.6 kB 28.4 MB/s eta 0:00:00
Installing collected packages: tzdata, numpy, pandas
Successfully installed numpy-2.1.2 pandas-2.2.3 tzdata-2024.2
```

In [14]:
```
#connection for pandas
pconn = ibm_db_dbi.Connection(conn)
```

In [15]:
```
#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]
```

```
/tmp/ipykernel_349/2373382735.py:5: UserWarning: pandas only supports SQLAlchemy con
nectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Ot
her DBAPI2 objects are not tested. Please consider using SQLAlchemy.
  pdf = pandas.read_sql(selectQuery, pconn)
```

Out[15]:
```
'Ahuja'
```

In [16]:
```
#print the entire data frame
pdf
```

Out[16]:

	ID	FNAME	LNAME	CITY	CCODE
0	1	Rav	Ahuja	MOOSETOWN	CA
1	2	Raul	Chong	Markham	CA
2	3	Hima	Vasudevan	Chicago	US

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

In [17]:
```
pdf.shape
```

Out[17]:
```
(3, 5)
```

Task 8: Close the Connection

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.

In [18]:
```
ibm_db.close(conn)
```

Out[18]:
```
True
```

Summary

In this tutorial you established a connection to a database instance of DB2 Warehouse on Cloud from a Python notebook using ibm_db API. Then created a table and insert a few rows of data into it. Then queried the data. You also retrieved the data into a pandas dataframe.

Author

Rav Ahuja


```
{toggle}##

{toggle}|

{toggle}|---|---|---|---|

{toggle}|
```

```
{toggle}
```

```
{toggle}
```