SQL Notes

- Origins IBM’s research labs.
- Early 1970’s.
- SQL (Structure Query Language).

What is Data?

- Facts (words, numbers).
- Pictures.
- One of the most critical assets of any business.

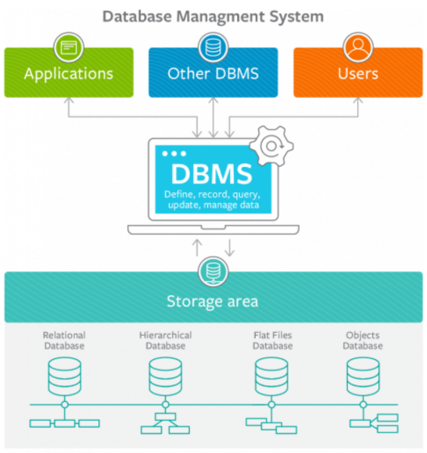
Database (DB)

- Wat is a database?
- A repository of data.
- Provides the functionality for adding, modifying, and querying that data.
- Different kinds of databases store data in different forms.
- Database.- A collection of integrated records (any collection of related information).
- Record.- A representation of a conceptual object.
- Database consists of data and metadata.
- DB’s can be stored in different ways.

Database Management Systems (DBMS)

- SQL is just a language, and there for special software needs to be used in order to perform queries.
- A query is a question you ask the database.
- DBMS.- Is a special software or program that helps users create and maintain a database.
- It makes it easy to manage large amounts of information.
- Handles security, backups, import/export of data.

DBMS Diagram



Types of Databases

- Relational Databases (SQL)
  - Organize data intone or more tables.
  - Each table has columns and rows.
  - A unique key identifies each row.
  - Data stored in tabular format.
  - Columns contain item properties.

- Non-Relational Databases (No SQL, not just SQL)
  -Organize data is anything but traditional table.
  - key: value stores.
  - Documents (JSON, XML, BLOB, etc.)
  - Flexible tables.
  - graphs, key-value hash, keys mapped into values.

Software

Relational Databases Management Systems (RDMMS)

- A set of software tools that control the data, access, organization, and storage.
- Helps users create and maintain a relational database.
- MySQL, Oracle, Postgre SQL, Maria DB, etc.
- Uses the structured query language.
- Used to perform CRUD (create, read, update, delete).
- Used to define tables and structures.
- SQL code is not always portable between applications without modification.

Non-Relational Databases Management Systems (NRDMMS)

- Helps users create and maintain a non-relational database.
- MongoDB, DynamoDB, apache Cassandra, firebase, etc.
- Implementation is specific.
- Any non-relational database falls under this category, so there is no set standard language.
- Most NRDMMS will implement their own language in order to perform CRUD, on the database.

Database Queries

- Queries are requests made to the database management system for specific information.
- As the database's structure becomes more and more complex, it becomes more difficult to retrieve specific pieces of information.
- Queries are instructions given to the RDBMS in SQL.

Database Service instances

- DBaaS provides users with access to database resources in cloud without setting no hardware and installing software.

Keys

- Relational databases always have a primary key, which can be the column id. Other keys are surrogate, natural, and foreign keys.
- Natural keys are used with the same purpose like in the real world.
- Foreign keys are links to other databases or tables and mark a relationship between databases and tables.

SQL

- SQL is a language used for interacting with relational databases management systems.
- You can use SQL to get the RDBMS to do things fir you. Like CRUD.
- Create, manage, design, administrate tasks, implements. etc.
- SQL implementations vary between systems.
- Concepts are the same, but implementation may vary.

SQL Types

- SQL can be referred as a hybrid language with 4 basic types of language
    1. Data Query Language (DQL) used to query the database for data.
    2. Data Definition Language (DDL) used for defining database schemas.
    3. Data Control Language (DCL) used for controlling access to the data in the database.
    4. Data Manipulation Language (DML) used for inserting, updating, and deleting data from the database.

SQL data types

Int	integer number.
Decimal (M,N)	float point number.
Varchar	string.
BLOB	binary language object.
Date	yyyy-mm-day
Timestamp	yyyy-mm-day and h:m:sec

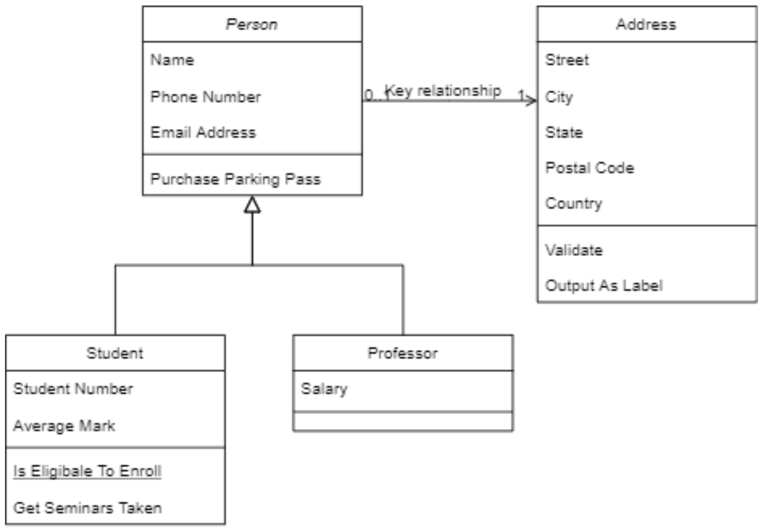
Basic SQL commands:

- Create a table
- Insert
- Select
- Update
- Delete

Information Model and Data Models

Relational Model

- Most used data model
- Allows for data independence.
- Data is stores in a table.
- Logical data independence -> physical data independence ->physical storage independence.
- Entity relationship data model, or ER data model.



- This is an Entity relationship diagram or ERD.
- Represents entities called tables and their relationships.

Entity Relationship models
- Used as a tool to design relational databases.

Mapping Entity Diagrams to tables
- Entities become tables
- Attributes are translated into columns

Primary and Foreign keys
- The primary key of a relational table uniquely identifies each table or row in a table, preventing duplication of data and providing a way of defining relations between tables.
- a PK is a constrain other constrains are not null values, which ensures that these fields cannot contain a NULL value.
- FK; Defined in other tables.
- Entities are independent objects which have attributes.
- Entities map to tables in a relational database.
- Attributes map to columns in a table.

Data Definition Language
- Statements
- Define, change, or drop data.

  Common DDL
  - CREATE, ALTER, TRUNCATE, DROP

Data Manipulation Language
- CRUD
- Read and modify data.

  Common DML
  - INSERT,SELECT,UPDATE,DELETE

Retrieving rows from a table

- Statement SELECT
- DML

Select statement: Query
Results from the query: Result set / table

```
SELECT * FROM <table_name>
```

- Retrieve just columns. The order of the columns displayed matches the order in the SELECT statement.

```
SELECT <column1>, <column2>, . . . FROM <table_name>
```

Limiting results

- WHERE clause
- restricts the result set
- Always requires a predicate
- Evaluates to a Boolean condition, True, False or Unknown.
- Used in the search condition of the where clause.

```
SELECT <column1>, <column2>,
FROM <table_name> WHERE <predicate>
```

Example

```
SELECT book_id, title
FROM book WHERE book_id = 'BI'
```

Use of Comparisons

Equal to	=
Greater than	>
Lesser than	<
Greater than or equal to	>=
Lesser than or equal to	<=
Not equal to	<> or !=

COUNT().

- A built-in function that retrieves the number of rows matching the query criteria.

```
SELECT COUNT(*) FROM <table_name>
```

DISTINCT()

- Used to remove duplicate values from a result set.
- Retrieve unique values in a column

```
SELECT DISTINCT <column> FROM <table_name>
```

LIMIT

- is used for restricting the rows retrieved from the database.
- retrieve just the first 10 rows in a table

```
SELECT * FROM <table_name> LIMIT 10
```

Insert statement

- Adding rows to a tale
- populate table with data

```
INSERT INTO <table_name>
(
 <col1>,<col4>,<col3>, ...)
VALUES (<val1>,<val2>,<val3>, ... )
```

Insert multiple rows

```
INSERT INTO <table_name>
(
  <col1>,<col4>,<col3>, ...)
  VALUES
 (<val1>,<val2>,<val3>, ... )
 (<val1>,<val2>,<val3>, ... )
```

Update and Delete

Altering rows of a table
used to read and modify data
Not specifying the WHERE clause updates all the selected values

```
UPDATE  <table_name>  SET
<column_name> = <values>
WHERE <condition>
```

Deleting rows from a table

```
DELETE FROM <table_name>
WHERE <condition>
```

String Patters

Use LIKE predicate with a string pattern for the search

```
WHERE <condition> LIKE <string_pattern>
```

`<string_pattern> -> %R` ; denotes any character that ends with R, can be placed before, after or both.
The % sign is called a wildcard character, and it is used to substitute other characters.

```
SELECT
  last_name,
  first_name,
     points ,
     (points+10)*100 as 'discount factor',
-- we can rename the column tag with as, for the result.
     points % 1
FROM customers;
```

Retrieving rows from a range or an interval

```
SELECT <column_name> FROM <table_name>
WHERE <predicate> BETWEEN <interval1> AND <interval2>
```

Using a set of values with the IN operator. The IN operator allows us to specify a set of values in a WHERE clause, it takes a list of expressions to compare against.

```
WHERE <column_name> IN (<val1>, <val2>)
```

Sorting Result sets
ORDER BY clause, is used to display the result set in alphabetical order

```
SELECT <column_name> FROM <table_name>
ORDER BY <column_name>
```

To sort in descending order use DESC at the end.

```
SELECT <column_name> FROM <table_name>
ORDER BY <column_name> DESC;
```

Grouping Results Sets
To eliminate duplicates, use.

```
SELECT DISCTINCT <column_name> FROM <table_name>;
```

Grouping clause, groups a result into subsets that has matching values for one or more columns.

```
SELECT <column_name> COUNT(<column_name>)
FROM <table_name> GROUP BY <column_name>;
```

Having Clause
Passing some conditions. It is used in combination with the GROUP BY clause.

```
SELECT <column_name> COUNT(<column_name>) AS Count
FROM <table_name> GROUP BY <new_column_name> <column_name>
HAVING COUNT(<column_name>) <number, string_pattern, etc.>;
```

Built-in Functions

Most databases come with built-in SQL functions. Built-in functions can significantly reduce the amount of data that needs to be retrieved. Can also speed up data processing

Aggregate on columns functions
Input.- Collection of values (e.g. entire column).
Output.- singe values.
Examples: SUM, MIN, MAX, AVG, etc.

SUM adds up all the values in a column. MIN,MAX, return the minimum and maximum value of a column. AVG, returns the average of a column.

Note.- Every column can be rewritten with a column alias, with the AS clause.

Scalar operations & Scalar and string functions
Scalar.- perform operations on every input values.
E.g., ROUND, LENGTH, UCASE, LCASE.

Note.- Aggregate functions cannot evaluate inside the WHERE clause.

Subqueries and Nested queries

Subqueries.- Is a query inside another query.

```
SELECT <column_name> FROM <table_name>
WHERE <column_name2> =
(
 SELECT MAX(<column_name2>)
 FROM <table_name>
);
```

The subqueries selections are done from inside out.
Why use subqueries? They are used to retrieve the list of a particular instance. For example, to retrieve the list of employees who earn more than the average salary.

```
SELECT Emp_ID, F_name, L_name, Salary FROM employees
WHERE Salary < (SELECT  AVG(Salary) FROM employees;
```

Subqueries in a list of columns

- Substitute column name with a subquery.
- This are called column expressions.

Example,

```
SELECT Emp_ID, Salary, (SELECT  AVG(Salary) FROM employees)
AS average_salary FROM employees;
```

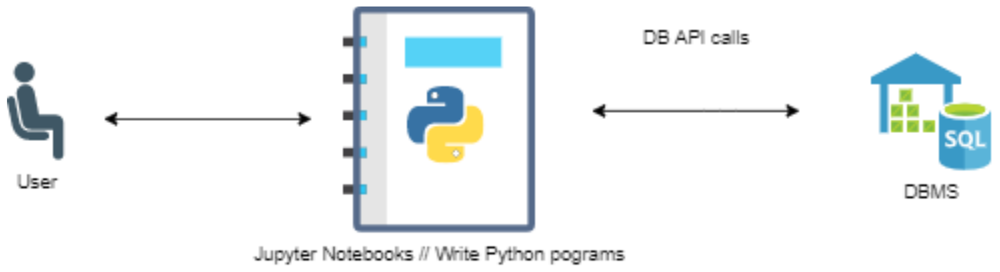
Subqueries in FROM clause

- Substitute the table name with a subquery.
- Called derived table or table expressions.

```
SELECT <col1>, <col2>, <col3> . . . FROM employees
AS <column_name>
```

Python standard API for Accessing relational databases

- Allows a single program to work with multiple kinds of relational databases.



Benefits of using DB-API

- Easy to implement and understand.
- Encourages similarity to access databases.
- achieves consistency.
- Portable across databases.
- Broad reach of databases connectivity from python.

Libraries used by database systems that connect to python apps

Database	DB API
IBM Db2	Ibm_db
Compose for MySQL	MySQL connector python
Compose for PostgreSQL	psy copg2
Compose for MongoDB	Py Mongo

Connection Objects

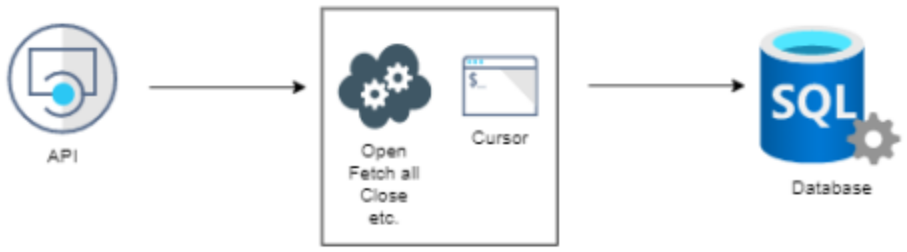
- Databases connections.
- Manage transactions.

Cursor objects

- Databases queries
- Scroll through result set.
- Retrieve results.

Connections methods

- .cursor() retrieve a new cursor object using connection.
- .commit() is used to commit any pending transaction on the database.
- .rollback() causes the database to roll back to the start of any pending transaction.
- .close() to close a database connection.
- These objects represent a database cursor which is used to manage the content of a fetch operation.



Example python code, that can be run in a Jupyter notebook

```python
from dmodule import connect

# create connection object
Connection = connect ('database_username', 'username','password')

# then create a cursor object to do queries and fetch results
Cursor = connection.cursor()

#run queries
Cursor.execute('SELECT * FROM my_table')
Results = cursor.fetchall()

#free resources, by closing all connections
Cursor.close()
Connection.close()
```

Connecting to a database with Ibm_db API

- The Ibm_db framework provides a variety of useful python functions for accessing and manipulating data on an IBM data server database, preparing and issuing SQL statements, fetching rows from result sets, calling stored procedures, committing and rolling back transactions, handling errors and retrieving metadata.
- Ibm_db API uses the IBM Data Server Driver for ODBC and CLI APIs to connect to IMB DB2 and Informix.
- Connecting to IBM DB2 requires connection credentials.

Creating DB connection credentials

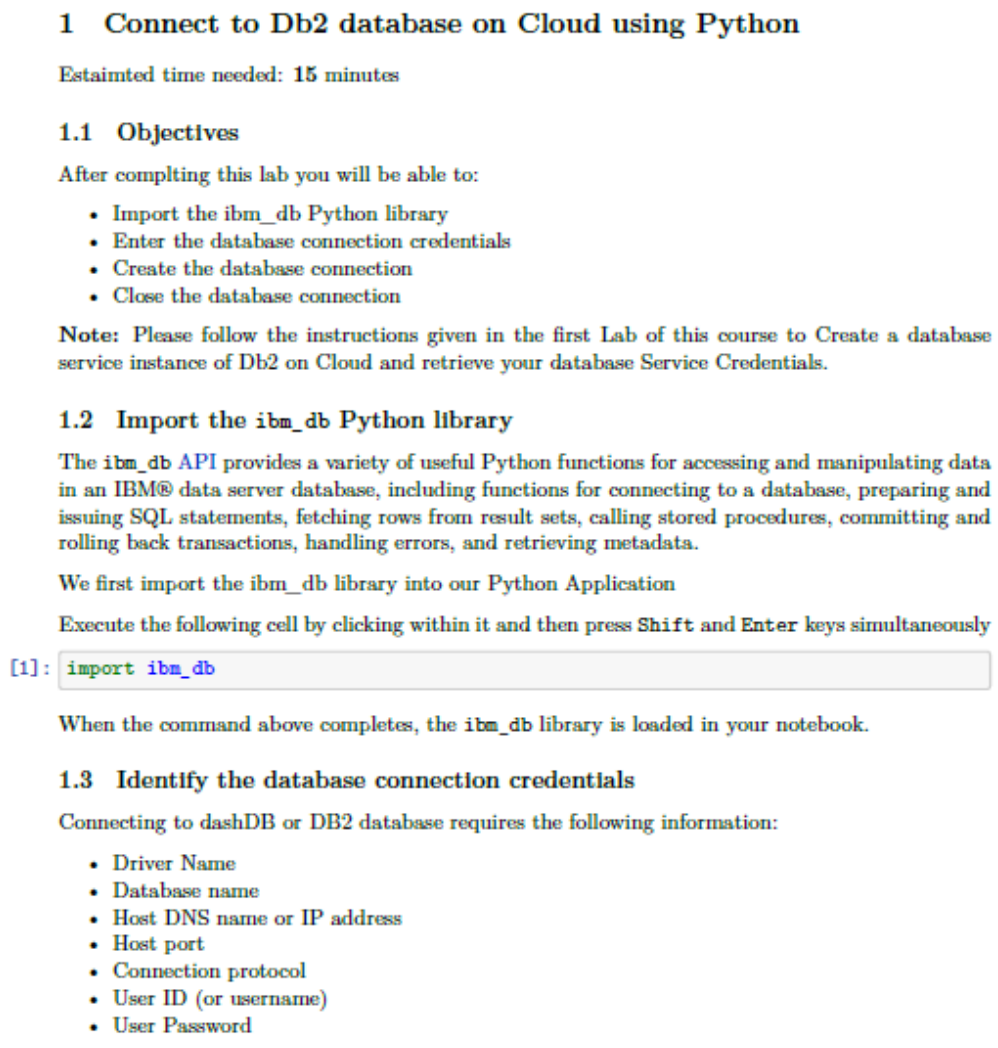
1) Log in into the IBM Cloud account.
2) Head to IBM Cloud resources dashboards.
3) Locate and click on the Db2 service listed under sources.
4) Click on Service Credentials.
5) and view the credentials for the remote connection.

Credentials have the following keys

- **port** is the database port
- **db** is the database name
- **host** is the hostname of the database instance
- **username** is the username you will use to connect
- **password** is the password you will use to connect
- (you may need to scroll down to see the password)
- **URI** (you will need this for Jupyter notebooks when using SQL Magic)

Example code for the credentials

```
{
  "db": "BLUDB",
  "dsn": "DATABASE=BLUDB;HOSTNAME=dashdb-txn-sbox-yp-dal09-
12.services.dal.bluemix.net;PORT=50000;PROTOCOL=TCPIP;UID=rdv33194;PWD=m^qgzq
r7m5llr2xd;",
  "host": "dashdb-txn-sbox-yp-dal09-12.services.dal.bluemix.net",
  "hostname": "dashdb-txn-sbox-yp-dal09-12.services.dal.bluemix.net",
  "https_url": "https://dashdb-txn-sbox-yp-dal09-
12.services.dal.bluemix.net",
  "jdbcurl": "jdbc:db2://dashdb-txn-sbox-yp-dal09-
12.services.dal.bluemix.net:50000/BLUDB",
  "parameters": {
    "role_crn": "crn:v1:bluemix:public:iam::::serviceRole:Manager"
  },
  "password": "m^qgzqr7m5llr2xd",
  "port": 50000,
  "ssldsn": "DATABASE=BLUDB;HOSTNAME=dashdb-txn-sbox-yp-dal09-
12.services.dal.bluemix.net;PORT=50001;PROTOCOL=TCPIP;UID=rdv33194;PWD=m^qgzq
r7m5llr2xd;Security=SSL;",
  "ssljdbcurl": "jdbc:db2://dashdb-txn-sbox-yp-dal09-
12.services.dal.bluemix.net:50001/BLUDB:sslConnection=true;",
  "uri": "db2://rdv33194:m%5Eqgzqr7m5llr2xd@dashdb-txn-sbox-yp-dal09-
12.services.dal.bluemix.net:50000/BLUDB",
  "username": "rdv33194"
}
```



1 Connect to Db2 database on Cloud using Python

Estaimted time needed: **15** minutes

1.1 Objectives

After complting this lab you will be able to:

- Import the ibm_db Python library
- Enter the database connection credentials
- Create the database connection
- Close the database connection

Note: Please follow the instructions given in the first Lab of this course to Create a database service instance of Db2 on Cloud and retrieve your database Service Credentials.

1.2 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

Execute the following cell by clicking within it and then press `Shift` and `Enter` keys simultaneously

```
[1]: import ibm_db
```

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

1.3 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 (or username)
- User Password

The images belong to the Lab-1_Ibm_Db2_python

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

Now enter your database credentials below and execute the cell with `Shift` + `Enter`

```
[9]: #Replace the placeholder values with your actual Db2 hostname, username, and␣
      ↪password:
     dsn_hostname = "dashdb-txn-sbox-yp-dal09-12.services.dal.bluemix.net" # e.g.:␣
      ↪"dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net"
     dsn_uid = "rdv33194"        # e.g. "abc12345"
     dsn_pwd = "m^qgzqr7m5llr2xd"      # 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"
```

1.4 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

```
[10]: #DO NOT MODIFY THIS CELL. 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)
```

```
DRIVER={IBM DB2 ODBC DRIVER};DATABASE=BLUDB;HOSTNAME=dashdb-txn-sbox-yp-dal09-12
.services.dal.bluemix.net;PORT=50000;PROTOCOL=TCPIP;UID=rdv33194;PWD=m^qgzqr7m51
1r2xd;
```

Now establish the connection to the database

```
[13]: #DO NOT MODIFY THIS CELL. Just RUN it with Shift + Enter
      #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() )
```

```
Connected to database:  BLUDB as user:  rdv33194 on host:  dashdb-txn-sbox-yp-
dal09-12.services.dal.bluemix.net
```

Congratulations if you were able to connect successfuly. Otherwise check the error and try again.

[14]:
```
#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)
```

```
DBMS_NAME:  DB2/LINUXX8664
DBMS_VER:   11.01.0404
DB_NAME:    BLUDB
```

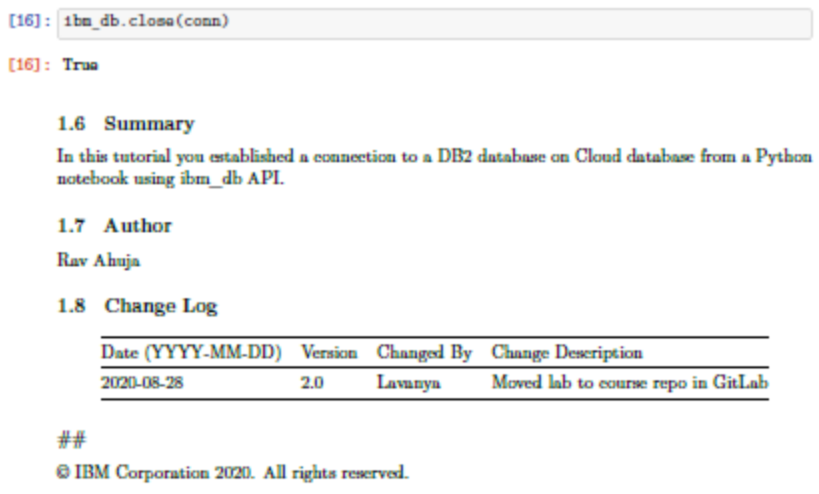
[15]:
```
#Retrieve Metadata for the Database Client / Driver
client = ibm_db.client_info(conn)

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)
```

```
DRIVER_NAME:           libdb2.a
DRIVER_VER:            11.05.0400
DATA_SOURCE_NAME:      BLUDB
DRIVER_ODBC_VER:       03.51
ODBC_VER:              03.01.0000
ODBC_SQL_CONFORMANCE:  EXTENDED
APPL_CODEPAGE:         1208
CONN_CODEPAGE:         1208
```

1.5 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.



```
[16]: ibm_db.close(conn)

[16]: True
```

1.6 Summary

In this tutorial you established a connection to a DB2 database on Cloud database from a Python notebook using ibm_db API.

1.7 Author

Rav Ahuja

1.8 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-08-28	2.0	Lavanya	Moved lab to course repo in GitLab

##

© IBM Corporation 2020. All rights reserved.

Creating Tables, Loading data and Querying data

- First we need to obtain a connection resource by connecting to the database using the method of the ibm_db api.
- To create a table, use the ibm_db.exec_immideate(), the parameters are, Connection, statement, options.

Example code:

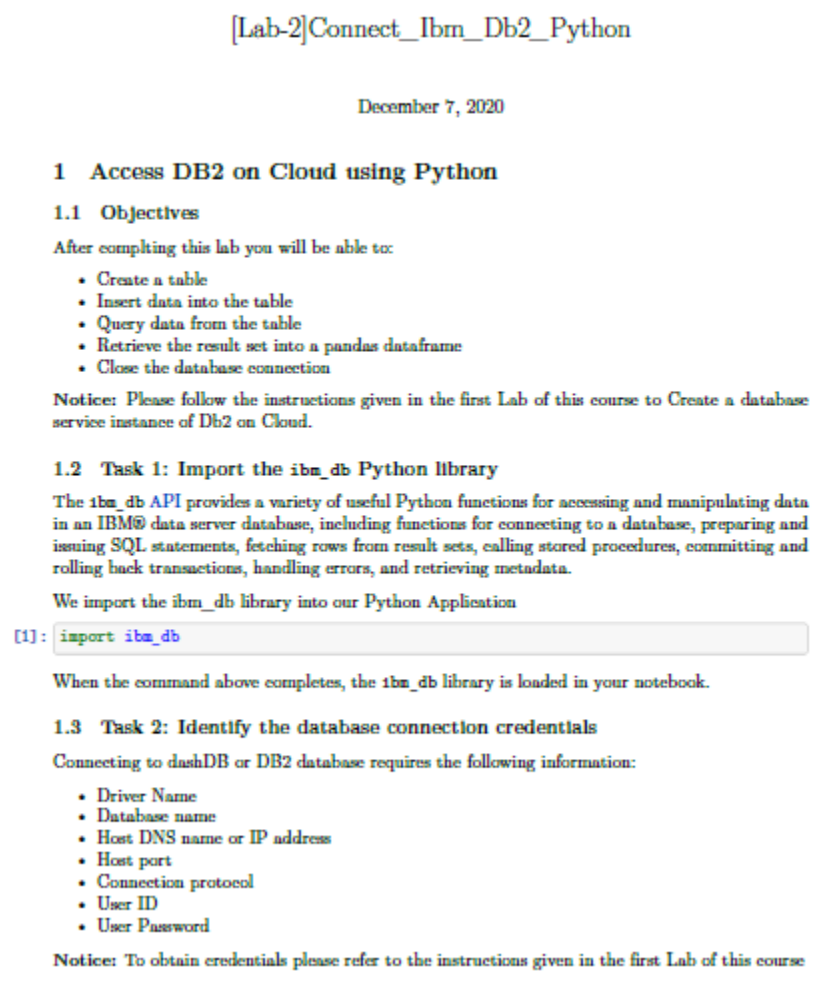
```
stmnt = ibm_db.exec_immediate('connection_symbol','SQL_statement')
```

The connection resource that was created is passed as the 1st parameter to the function, the next parameter is the SQL code statement, which is a query.

Using pandas

Data can be loaded into a data frame; the data frame represents a tabular spreadsheet format data structure containing an ordered collection of columns.

The next Jupyter notebook belongs to the Lab-2



[Lab-2]Connect_Ibm_Db2_Python

December 7, 2020

1 Access DB2 on Cloud using Python

1.1 Objectives

After complting 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.

1.2 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

```
[1]: import ibm_db
```

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

1.3 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"

```
[2]: #Replace the placeholder values with the actuals for your Db2 Service␣
      ↪Credentials
     dsn_driver = "{IBM DB2 ODBC DRIVER}"
     dsn_database = "BLUDB"            # e.g. "BLUDB"
     dsn_hostname = "dashdb-txn-sbox-yp-dal09-12.services.dal.bluemix.net"      # e.
      ↪g.: "dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net"
     dsn_port = "50000"                # e.g. "50000"
     dsn_protocol = "TCPIP"            # i.e. "TCPIP"
     dsn_uid = "rdv33194"              # e.g. "abc12345"
     dsn_pwd = "m^qgzqr7m511r2xd"              # e.g. "7dBZ3wWt9XN6$o0J"
```

1.4 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

```
[3]: #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};").format(dsn_driver, dsn_database, dsn_hostname, dsn_port,␣
      ↪dsn_protocol, dsn_uid, dsn_pwd)

     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() )
```

```
Connected to database:  BLUDB as user:  rdv33194 on host:  dashdb-txn-sbox-yp-
dal09-12.services.dal.bluemix.net
```

1.5 Task 4: Create a table in the database

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

[4]:
```
#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)
```

1.6 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

[8]:
```
#Construct the Create Table DDL statement - replace the ... with rest of the␣
↪statement
createQuery = "create table INSTRUCTOR(id INTEGER PRIMARY KEY NOT NULL, FNAME␣
↪VARCHAR(20) ,LNAME VARCHAR(20), CITY VARCHAR(20), CCODE CHAR(2))"

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

Double-click **here** for the solution.

1.7 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

[9]:
```
#Construct the query - replace ... with the insert statement
insertQuery = "INSERT INTO INSTRUCTOR (id,FNAME,LNAME,CITY,CCODE) VALUES␣
↪(1,'Rav','Ahuja','TORONTO','CA')"

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

Double-click **here** for the solution.

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

[10]:
```
#replace ... with the insert statement that inerts the remaining two rows of␣
↪data
insertQuery2 = "INSERT INTO INSTRUCTOR (id,FNAME,LNAME,CITY,CCODE) VALUES␣
↪(2,'Raul','Chong','Markham','CA'),(3,'Hima','Vasudevan','Chicago','US')"

#execute the statement
```

```
insertStmt2 = ibm_db.exec_immediate(conn, insertQuery2)
```

Double-click here for the solution.

1.8 Task 6: Query data in the table

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

[13]:
```
#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)
```

[13]:
```
{'ID': 1,
 0: 1,
 'FNAME': 'Rav',
 1: 'Rav',
 'LNAME': 'Ahuja',
 2: 'Ahuja',
 'CITY': 'TORONTO',
 3: 'TORONTO',
 'CCODE': 'CA',
 4: 'CA'}
```

Double-click here for the solution.

[14]:
```
#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"))
```

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

Double-click here for the solution.

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

[24]:
```
#update ... with the upate statement
updateQuery = "UPDATE INSTRUCTOR SET  CITY= 'MOOSETOWN' WHERE FNAME='Rav'"

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

Double-click here for the solution.

1.9 Task 7: Retrieve data into Pandas

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

```
[16]: import pandas
      import ibm_db_dbi
```

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

```
[26]: #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]
```

```
[26]: 'Ahuja'
```

```
[27]: #print the entire data frame
      pdf
```

```
[27]:    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

```
[28]: pdf.shape
```

```
[28]: (3, 5)
```

1.10 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.

```
[29]: ibm_db.close(conn)
```

```
[29]: True
```

1.11 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.

1.12 Author

IBM SQL_Course

1.13 Modify By

The Guy

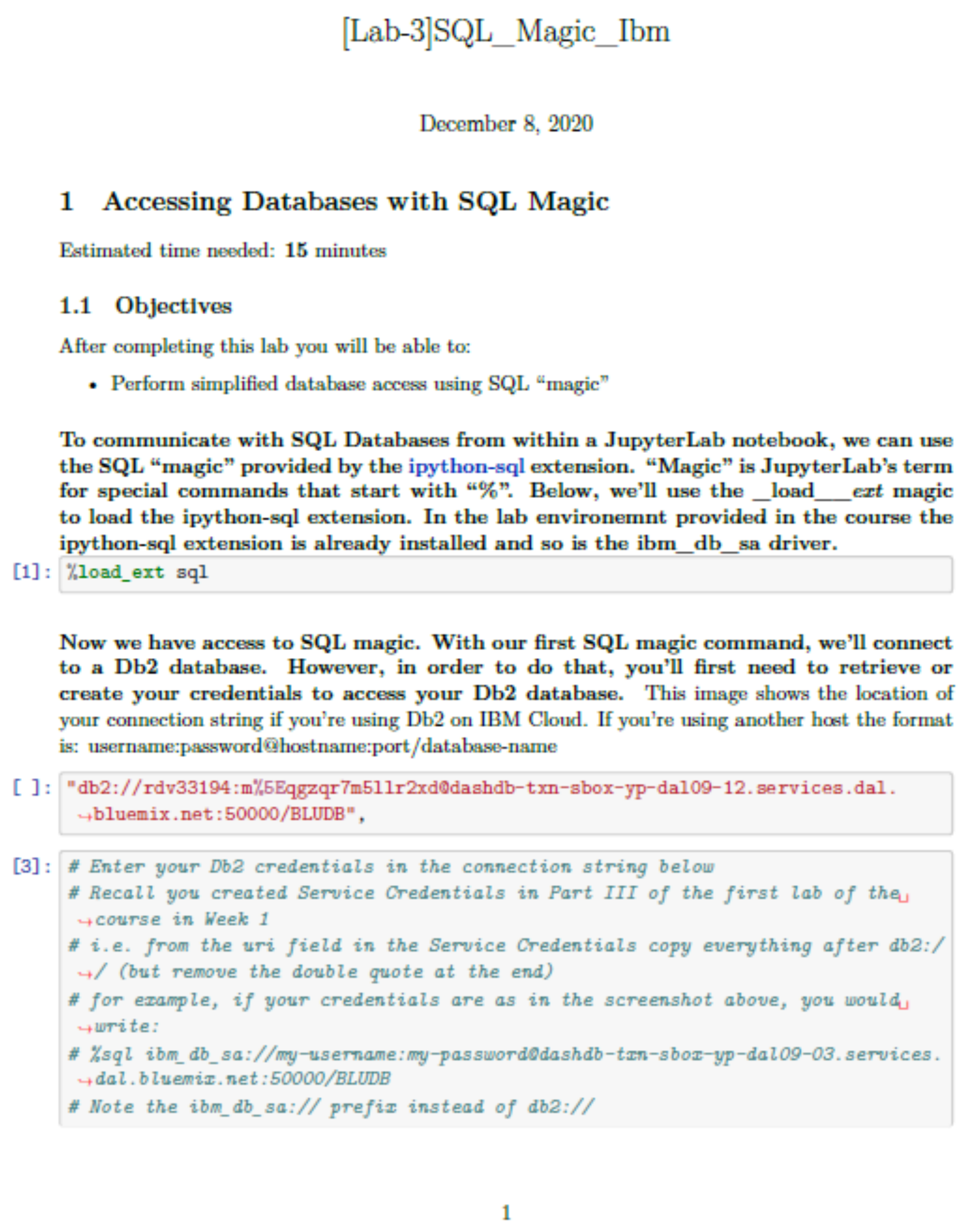
1.14 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-08-28	2.0	THe Guy	Moved lab to course repo in GitLab

##

```
[ ]:
```

Lab 3 Accessing the IBM Db2 database with SQL magic



[Lab-3]SQL_Magic_Ibm

December 8, 2020

1 Accessing Databases with SQL Magic

Estimated time needed: **15** minutes

1.1 Objectives

After completing this lab you will be able to:

- Perform simplified database access using SQL "magic"

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.

[1]:
```
%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. 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

[ ]:
```
"db2://rdv33194:m%5Eqgzqr7m5llr2xd@dashdb-txn-sbox-yp-dal09-12.services.dal.
↪bluemix.net:50000/BLUDB",
```

[3]:
```
# 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://
```

1

```
# 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://rdv33194:m%5Eqgzqr7m5llr2xd@dashdb-txn-sbox-yp-dal09-12.
↪services.dal.bluemix.net:50000/BLUDB
```

[3]: 'Connected: rdv33194@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.

[4]:
```
%%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),
('Greece', 'Aharon', 'Tunnow', 48),
('Russia', 'Bail', 'Goodwin', 46),
('Poland', 'Cole', 'Winteringham', 49),
('Sweden', 'Emlyn', 'Erricker', 55),
('Russia', 'Cathee', 'Sivewright', 49),
('China', 'Barny', 'Ingerson', 57),
('Uganda', 'Sharla', 'Papaccio', 55),
('China', 'Stella', 'Youens', 51),
('Poland', 'Julio', 'Buesden', 48),
('United States', 'Tiffie', 'Cosely', 58),
('Poland', 'Auroora', 'Stiffell', 45),
('China', 'Clarita', 'Huet', 52),
('Poland', 'Shannon', 'Goulden', 45),
('Philippines', 'Emylee', 'Privost', 50),
('France', 'Madelina', 'Burk', 49),
('China', 'Saunderson', 'Root', 58),
('Indonesia', 'Bo', 'Waring', 55),
('China', 'Hollis', 'Domotor', 45),
('Russia', 'Robbie', 'Collip', 46),
('Philippines', 'Davon', 'Donisi', 46),
('China', 'Cristabel', 'Radeliffe', 48),
('China', 'Wallis', 'Bartleet', 58),
('Moldova', 'Arleen', 'Stailey', 38),
('Ireland', 'Mendel', 'Grumble', 58),
('China', 'Sallyann', 'Exley', 51),
('Mexico', 'Kain', 'Swaite', 46),
('Indonesia', 'Alonso', 'Bulteel', 45),
('Armenia', 'Anatol', 'Tankus', 51),
('Indonesia', 'Coralyn', 'Dawkins', 48),
('China', 'Deanne', 'Edwinson', 45),
('China', 'Georgiana', 'Epple', 51),
('Portugal', 'Bartlet', 'Breese', 56),
('Azerbaijan', 'Idalina', 'Lukash', 50),
('France', 'Livvie', 'Flory', 54),
('Malaysia', 'Nonie', 'Borit', 48),
('Indonesia', 'Clio', 'Mugg', 47),
('Brazil', 'Westley', 'Measor', 48),
('Philippines', 'Katrinka', 'Sibbert', 51),
('Poland', 'Valentia', 'Mounch', 50),
('Norway', 'Sheilah', 'Hedditch', 53),
('Papua New Guinea', 'Itch', 'Jubb', 50),
('Latvia', 'Stesha', 'Garnson', 53),
('Canada', 'Cristionna', 'Wadmore', 46),
('China', 'Lianna', 'Gatward', 43),
('Guatemala', 'Tanney', 'Vials', 48),
('France', 'Alma', 'Zavittieri', 44),
('China', 'Alvira', 'Tamas', 50),
('United States', 'Shanon', 'Peres', 45),
('Sweden', 'Maisey', 'Lynas', 53),
('Indonesia', 'Kip', 'Hothersall', 46),
('China', 'Cash', 'Landis', 48),
('Panama', 'Kennith', 'Digance', 45),
('China', 'Ulberto', 'Riggeard', 48),
('Switzerland', 'Judy', 'Gilligan', 49),
('Philippines', 'Tod', 'Trevaskus', 52),
('Brazil', 'Herold', 'Heggs', 44),
('Latvia', 'Verney', 'Note', 50),
('Poland', 'Temp', 'Ribey', 50),
('China', 'Conroy', 'Egdal', 48),
('Japan', 'Gabie', 'Alessandone', 47),
('Ukraine', 'Devlen', 'Chaperlin', 54),
('France', 'Babbette', 'Turner', 51),
('Czech Republic', 'Virgil', 'Scotney', 52),
('Tajikistan', 'Zorina', 'Bedow', 49),
('China', 'Aidan', 'Rudeyeard', 50),
('Ireland', 'Saunder', 'MacLice', 48),
('France', 'Waly', 'Brunstan', 53),
('China', 'Gisele', 'Enns', 52),
('Peru', 'Mina', 'Winchester', 48),
('Japan', 'Torie', 'MacShirrie', 50),
('Russia', 'Benjamen', 'Kenford', 51),
('China', 'Etan', 'Burn', 53),
('Russia', 'Merralee', 'Chaperlin', 38),
('Indonesia', 'Lanny', 'Malam', 49),
('Canada', 'Wilhelm', 'Deeprose', 54),
('Czech Republic', 'Lari', 'Hillhouse', 48),
('China', 'Ossie', 'Woodley', 52),
('Macedonia', 'April', 'Tyer', 50),
('Vietnam', 'Madelon', 'Dansey', 53),
('Ukraine', 'Korella', 'McNamee', 52),
('Jamaica', 'Linnea', 'Cannam', 43),
('China', 'Mart', 'Coling', 52),
('Indonesia', 'Marna', 'Causbey', 47),
('China', 'Berni', 'Daintier', 55),
('Poland', 'Cynthia', 'Hassell', 49),
('Canada', 'Carna', 'Schule', 49),
('Indonesia', 'Malia', 'Blight', 48),
('China', 'Paulo', 'Seivertsen', 47),
('Niger', 'Kaylee', 'Hearley', 54),
('Japan', 'Maure', 'Jandak', 46),
('Argentina', 'Foss', 'Feavers', 45),
('Venezuela', 'Ron', 'Leggitt', 60),
('Russia', 'Flint', 'Gokes', 40),
('China', 'Linet', 'Conelly', 52),
('Philippines', 'Nikolas', 'Birtwell', 57),
('Australia', 'Eduard', 'Leipelt', 53)
```

```
 * ibm_db_sa://rdv33194:***@dashdb-txn-sbox-yp-
dal09-12.services.dal.bluemix.net:50000/BLUDB
Done.
99 rows affected.
```

[4]:
```
[]
```

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.

[5]:
```
country = "Canada"
%sql select * from INTERNATIONAL_STUDENT_TEST_SCORES where country = :country
```

```
 * ibm_db_sa://rdv33194:***@dashdb-txn-sbox-yp-
dal09-12.services.dal.bluemix.net:50000/BLUDB
Done.
```

```
[5]: [('Canada', 'Cristionna', 'Wadmore', 46),
      ('Canada', 'Wilhelm', 'Deeprose', 54),
      ('Canada', 'Carma', 'Schule', 49)]
```

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.

```
[6]: 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
```

```
 * ibm_db_sa://rdv33194:***@dashdb-txn-sbox-yp-
dal09-12.services.dal.bluemix.net:50000/BLUDB
Done.
```

```
[6]: [(38, Decimal('2')),
      (40, Decimal('1')),
      (43, Decimal('2')),
      (44, Decimal('2')),
      (45, Decimal('8')),
      (46, Decimal('7')),
      (47, Decimal('4')),
      (48, Decimal('14')),
      (49, Decimal('8')),
      (50, Decimal('10')),
      (51, Decimal('8')),
      (52, Decimal('8')),
      (53, Decimal('8')),
      (54, Decimal('5')),
      (55, Decimal('4')),
      (56, Decimal('1')),
      (57, Decimal('2')),
      (58, Decimal('4')),
      (60, Decimal('1'))]
```

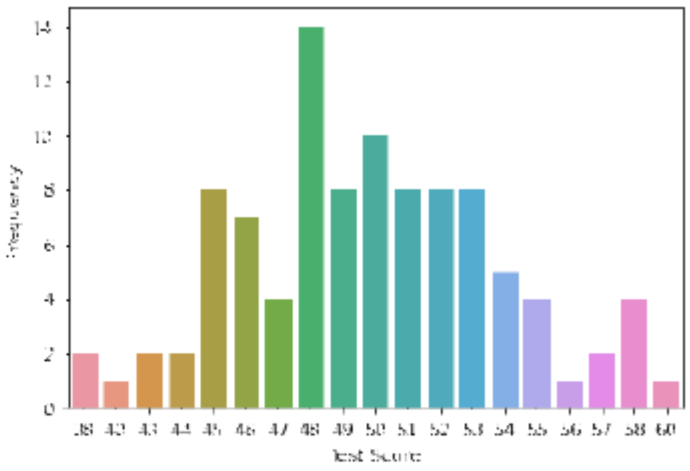
Converting Query Results to DataFrames

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.

```
[7]: 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"!

```
[8]: %%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;
```

```
 * ibm_db_sa://rdv33194:***@dashdb-txn-sbox-yp-
dal09-12.services.dal.bluemix.net:50000/BLUDB
```

```
    Done.
```

```
[8]: [('United States', 'Marshall', 'Bernadot', 54),
      ('Ghana', 'Celinda', 'Malkin', 51),
      ('Ukraine', 'Guillermo', 'Furze', 53),
      ('Greece', 'Aharon', 'Tunnow', 48),
      ('Russia', 'Bail', 'Goodwin', 46),
      ('Poland', 'Cole', 'Winteringham', 49),
      ('Sweden', 'Emlyn', 'Erricker', 55),
      ('Russia', 'Cathee', 'Sivewright', 49),
      ('China', 'Barny', 'Ingerson', 57),
      ('Uganda', 'Sharla', 'Papaccio', 55),
      ('China', 'Stella', 'Youens', 51),
      ('Poland', 'Julio', 'Buesden', 48),
      ('United States', 'Tiffie', 'Cosely', 58),
      ('Poland', 'Auroora', 'Stiffell', 45),
      ('China', 'Clarita', 'Huet', 52),
      ('Poland', 'Shannon', 'Goulden', 45),
      ('Philippines', 'Emylee', 'Privost', 50),
      ('France', 'Madelina', 'Burk', 49),
      ('China', 'Saunderson', 'Root', 58),
      ('Indonesia', 'Bo', 'Waring', 55),
      ('China', 'Hollis', 'Domotor', 45),
      ('Russia', 'Robbie', 'Collip', 46),
      ('Philippines', 'Davon', 'Donisi', 46),
      ('China', 'Cristabel', 'Radeliffe', 48),
      ('China', 'Wallis', 'Bartleet', 58),
      ('Moldova', 'Arleen', 'Stailey', 38),
      ('Ireland', 'Mendel', 'Grumble', 58),
      ('China', 'Sallyann', 'Exley', 51),
      ('Mexico', 'Kain', 'Swaite', 46),
      ('Indonesia', 'Alonso', 'Bulteel', 45),
      ('Armenia', 'Anatol', 'Tankus', 51),
      ('Indonesia', 'Coralyn', 'Dawkins', 48),
      ('China', 'Deanne', 'Edwinson', 45),
      ('China', 'Georgiana', 'Epple', 51),
      ('Portugal', 'Bartlet', 'Breese', 56),
      ('Azerbaijan', 'Idalina', 'Lukash', 50),
      ('France', 'Livvie', 'Flory', 54),
      ('Malaysia', 'Nonie', 'Borit', 48),
      ('Indonesia', 'Clio', 'Mugg', 47),
      ('Brazil', 'Westley', 'Measor', 48),
      ('Philippines', 'Katrinka', 'Sibbert', 51),
      ('Poland', 'Valentia', 'Mounch', 50),
      ('Norway', 'Sheilah', 'Hedditch', 53),
      ('Papua New Guinea', 'Itch', 'Jubb', 50),
      ('Latvia', 'Stesha', 'Garnson', 53),
('Canada', 'Cristionna', 'Wadmore', 46),
('China', 'Lianna', 'Gatward', 43),
('Guatemala', 'Tanney', 'Vials', 48),
('France', 'Alma', 'Zavittieri', 44),
('China', 'Alvira', 'Tamas', 50),
('United States', 'Shanon', 'Peres', 45),
('Sweden', 'Maisey', 'Lynas', 53),
('Indonesia', 'Kip', 'Hothersall', 46),
('China', 'Cash', 'Landis', 48),
('Panama', 'Kennith', 'Digance', 45),
('China', 'Ulberto', 'Riggeard', 48),
('Switzerland', 'Judy', 'Gilligan', 49),
('Philippines', 'Tod', 'Trevaskus', 52),
('Brazil', 'Herold', 'Heggs', 44),
('Latvia', 'Verney', 'Note', 50),
('Poland', 'Temp', 'Ribey', 50),
('China', 'Conroy', 'Egdal', 48),
('Japan', 'Gabie', 'Alessandone', 47),
('Ukraine', 'Devlen', 'Chaperlin', 54),
('France', 'Babbette', 'Turner', 51),
('Czech Republic', 'Virgil', 'Scotney', 52),
('Tajikistan', 'Zorina', 'Bedow', 49),
('China', 'Aidan', 'Rudeyeard', 50),
('Ireland', 'Saunder', 'MacLice', 48),
('France', 'Waly', 'Brunstan', 53),
('China', 'Gisele', 'Enns', 52),
('Peru', 'Mina', 'Winchester', 48),
('Japan', 'Torie', 'MacShirrie', 50),
('Russia', 'Benjamen', 'Kenford', 51),
('China', 'Etan', 'Burn', 53),
('Russia', 'Merralee', 'Chaperlin', 38),
('Indonesia', 'Lanny', 'Malam', 49),
('Canada', 'Wilhelm', 'Deeprose', 54),
('Czech Republic', 'Lari', 'Hillhouse', 48),
('China', 'Ossie', 'Woodley', 52),
('Macedonia', 'April', 'Tyer', 50),
('Vietnam', 'Madelon', 'Dansey', 53),
('Ukraine', 'Korella', 'McNamee', 52),
('Jamaica', 'Linnea', 'Cannam', 43),
('China', 'Mart', 'Coling', 52),
('Indonesia', 'Marna', 'Causbey', 47),
('China', 'Berni', 'Daintier', 55),
('Poland', 'Cynthia', 'Hassell', 49),
('Canada', 'Carma', 'Schule', 49),
('Indonesia', 'Malia', 'Blight', 48),
('China', 'Paulo', 'Seivertsen', 47),
('Niger', 'Kaylee', 'Hearley', 54),
('Japan', 'Maure', 'Jandak', 46),
('Argentina', 'Foss', 'Feavers', 45),
('Venezuela', 'Ron', 'Leggitt', 60),
('Russia', 'Flint', 'Gokes', 40),
('China', 'Linet', 'Conelly', 52),
('Philippines', 'Nikolas', 'Birtwell', 57),
('Australia', 'Eduard', 'Leipelt', 53)]
```

1.2 Author

IBM_SQL Course

1.3 Modify by

The Guy

1.4 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-07-17	2.0	The Guy	Moved lab to course repo in GitLab

##

```
[ ]:
```