**SQL- Structured Query Language**

In this course you will learn about:

* The fundamentals of relational databases and basic SQL commands that you can use to create, manage and query them.
* More advanced SQL commands that enable you to group and sort the results of queries, use built-in functions, and include results from multiple tables.
* Accessing databases programmatically with Python using Jupyter Notebooks

Syllabus

* Module 1 - Introduction to Databases
* Module 2 - Basic SQL
* Module 3 - String Patterns, Ranges, Sorting, and Grouping
* Module 4 - Functions, Sub-Queries, Multiple Tables
* Module 5 - Accessing Databases using Python
* Module 6 - Course Assignment

In Module 1

You will be introduced to databases and relational concepts. You will also create a database instance on the cloud.

Installation of IBM Db2 cloud-<https://www.youtube.com/watch?v=k1Wj2Sc5Ing>

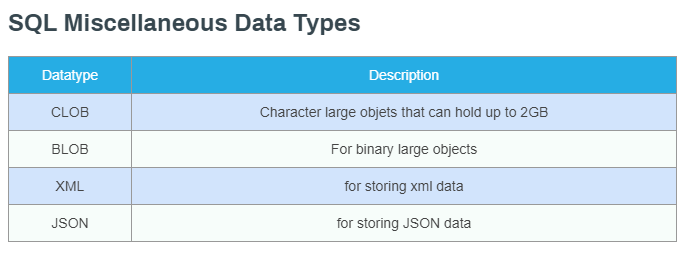
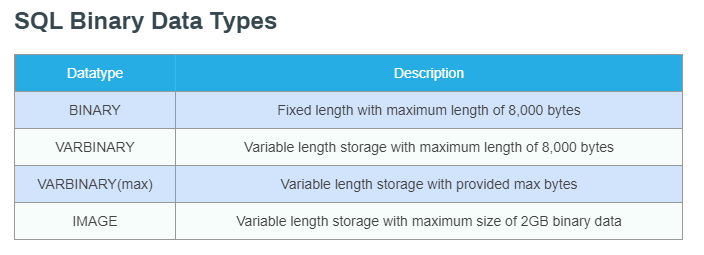
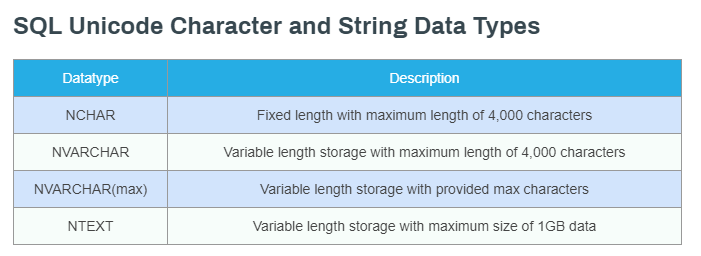
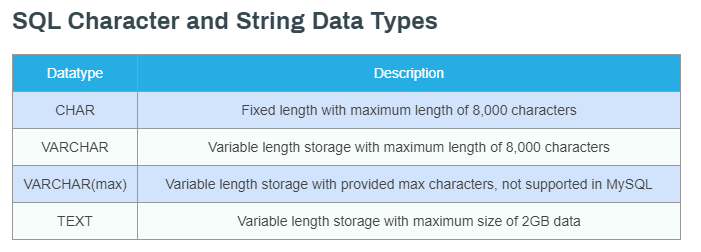
In Module 2

You will some of the basic SQL statements. You will also write and practice basic SQL hands-on on a live database.

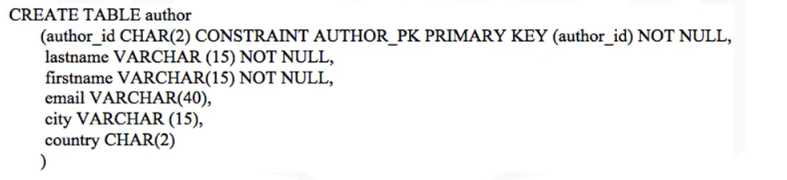
Relational Database-Data stored in tabular form

Data Types-





Creating table-



Here author\_id is assign as primary key which defines it should not repeated in dataset.

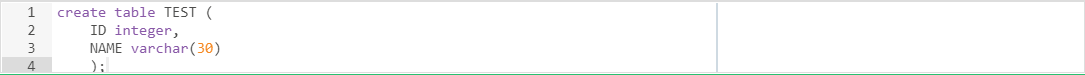
**Examples to CREATE and DROP tables-**

Here we will look at some examples to create and drop tables.

In the previous video we saw the general syntax to create a table:

create table TABLENAME (  
   COLUMN1 datatype,  
   COLUMN2 datatype,  
   COLUMN3 datatype,  
   ...  
);

Therefore to create a table called TEST with two columns - ID of type integer, and Name of type varchar, we could create it using the following SQL statement:



Now let's create a table called COUNTRY with an ID column, a two letter country code column, and a variable length country name column:



Sometimes you may see additional keywords in a create table statement:



In the above example the ID column has the "NOT NULL" constraint added after the datatype - meaning that it cannot contain a NULL or an empty value. If you look at the last row in the create table statement above you will note that we are using ID as a Primary Key and the database does not allow Primary Keys to have NULL values. A Primary Key is a unique identifier in a table, and using Primary Keys can help speed up your queries significantly.

If the table you are trying to create already exists in the database, you will get an error indicating "table XXX.YYY already exists". To circumvent this error, either create a table with a different name or first DROP the existing table. It is quite common to issue a DROP before doing a CREATE in test and development scenarios.

Here is an example:



WARNING: before dropping a table ensure that it doesn't contain important data that can't be recovered easily.

Note that if the table does not already exist and you try to drop it, you will see an error like "XXX.YYY is an undefined name". You can ignore this error as long as the subsequent CREATE statement executed successfully.

Types of statement-

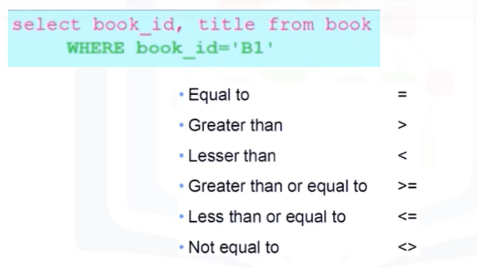
* DDL-Data Definition language statement-Define, change or drop data.
* DML-Data Manipulation language Statement-Read & modify data.

SELECT Statement-

Used to see data from database.It is DML Statement.

i.e., select \* from <tablename>

Where Clause-



**SELECT statement examples**

The general syntax of SELECT statement is:

select COLUMN1, COLUMN2, ... from TABLE1 ;

To retrieve all columns from the COUNTRY table we could use "\*" instead of specifying individual column names:

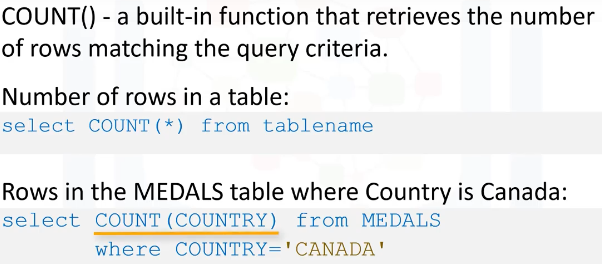
select \* from COUNTRY ;

The WHERE clause can be added to your query to filter results or get specific rows of data. To retrieve data for all rows in the COUNTRY table where the ID is less than 5:

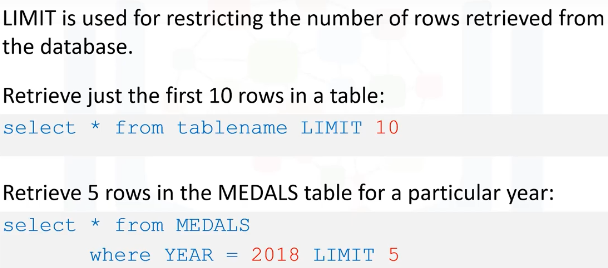
select \* from COUNTRY where ID < 5 ;

In case of character based columns the values of the predicates in the where clause need to be enclosed in single quotes. To retrieve the data for the country with country code "CA" we would issue:

select \* from COUNTRY where CCODE = 'CA';



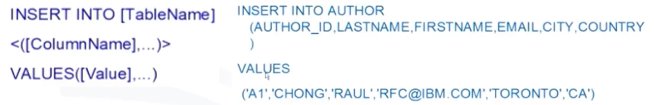




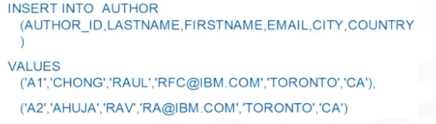
INSERT statement-

After creating a table we have to insert a data with INSERT statement.

INSERT statement is DML statement.

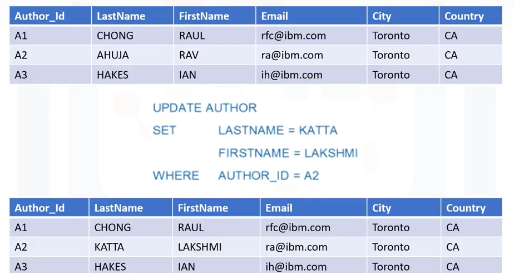


Inserting multiple rows-



UPDATE statement-(DML Statement)

After inserting data if we have to alter it we have to use UPDATE statement-



DELETE statement-(DML Statement)



**Composing and Running Basic SQL Queries-**

So far in this module you learned about the five basic SQL statements to create tables, insert data, select results, update, and delete data. In this lab, you will practice composing and running these statements via hands-on experiences.

To write and execute the SQL statements in this lab, you will need to launch the SQL editor (Run SQL) in your Db2 database instance on IBM Cloud that you created in the previous lab.

NOTE: Some users have reported issues performing this lab using Microsoft Edge. It is recommended using Google Chrome or Mozilla Firefox as your web browser to complete these labs.

First you will launch the SQL editor in the Db2 console using the following steps:

i) Go to your IBM Cloud dashboard (you may need to log into IBM Cloud in the process):

<https://cloud.ibm.com/resources>

ii) Expand the Cloud Foundry Services and locate and click on your instance of Db2 you provisioned in the previous lab (the name typically starts with Db2-xx for example Db2-fk, Db2-50, etc.)

**If getting issue while developing connection between Jupiter notebook & sql, please go through-**

<https://stackoverflow.com/questions/56098681/how-to-fix-sqlalchemy-connection-problem-connection-info-needed-in-sqlalchemy>

Connection of Jupiter notebook with ibm sql

import ibm\_db

import ibm\_db\_sa

import sqlalchemy

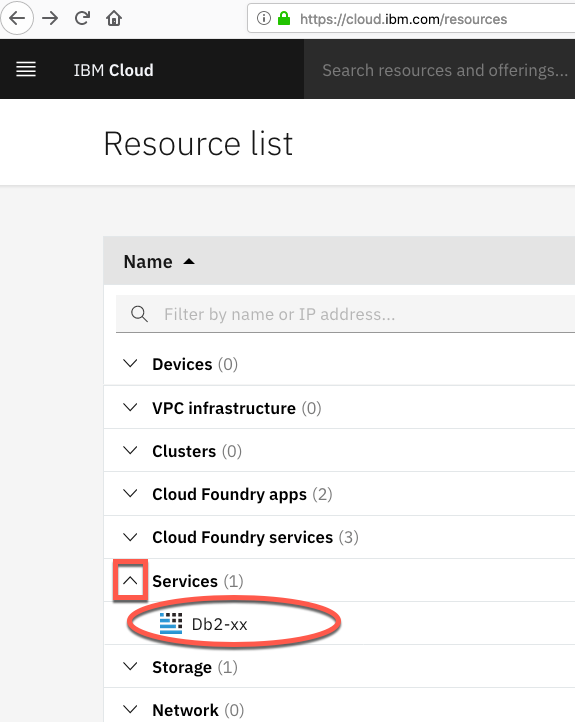
from sqlalchemy import \*

%load\_ext sql

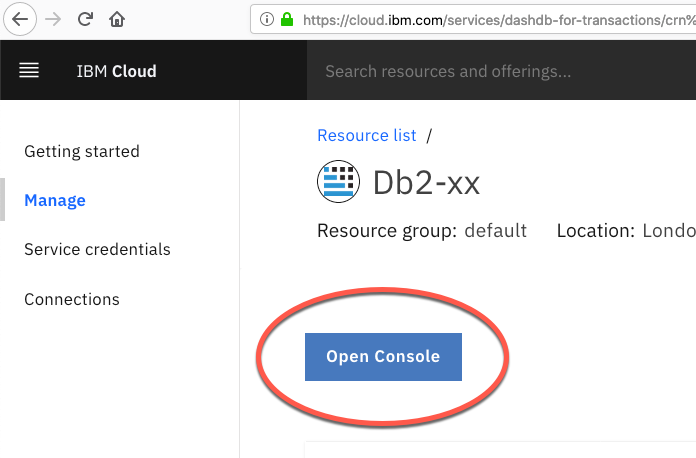
%sql ibm\_db\_sa://rcm78049:mxtzzbnh%40svr0rng@dashdb-txn-sbox-yp-lon02-04.services.eu-gb.bluemix.net:50000/BLUDB

Green highlighted in connection url which we can retrieve from db module on ibm cloud.

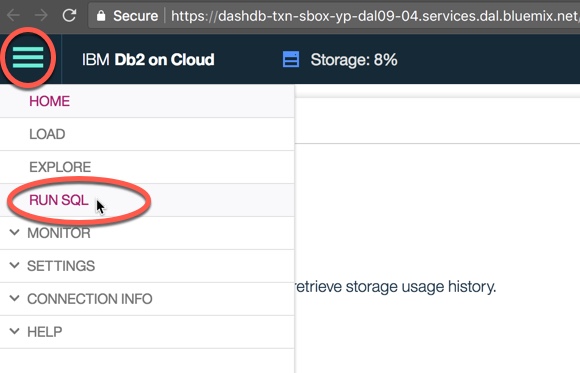
If any module is not there just type !pip install module name in Jupiter notebook.



iii) Click on the **Open Console** button.



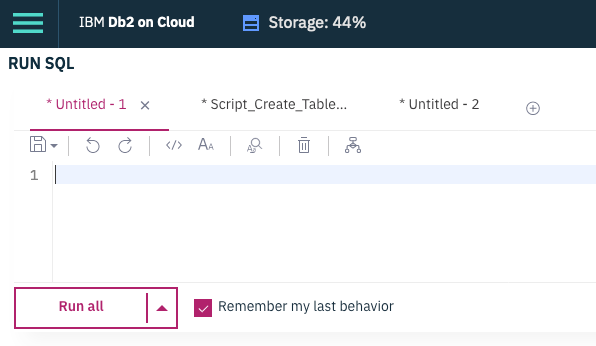
iv) The Db2 console will open in a new tab in your web browser. Click on the 3 bar menu icon in the top left corner and then click on RUN SQL.



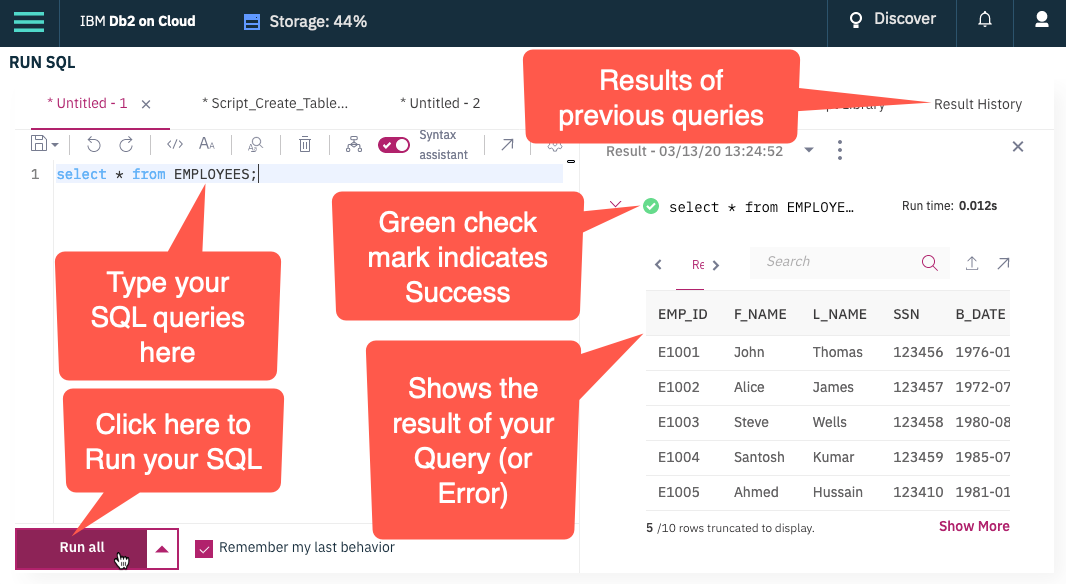
On the next screen click on "Blank"



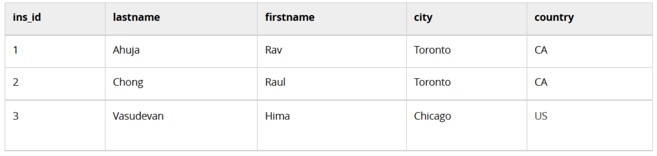
v) The SQL editor will open where you can start typing and running queries.



The SQL editor has several areas for different things:



Consider the following table called **INSTRUCTOR**:

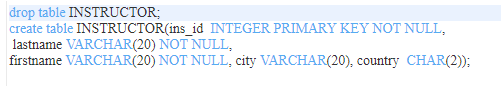
****

Task 0: Drop the table INSTRUCTOR from the database in case it already exists, so that we start from a clean state.

(Hint: Ignore the undefined error if this table does not already exist in your database)

Task 1: Create the INSTRUCTOR table as defined above. Have the ins\_id be the primary key, and ensure the lastname and firstname are not null.

(Hint: ins\_id is of type INTEGER, country of type CHAR(2), and rest of the fields VARCHAR)



Task 2A: Insert one row into the INSTRUCTOR table for the the instructor Rav Ahuja.

(Hint: values for the character fields require a single quotation mark (') before and after each value)

C:\Users\pares\Desktop\SQL(EDX)\16.png

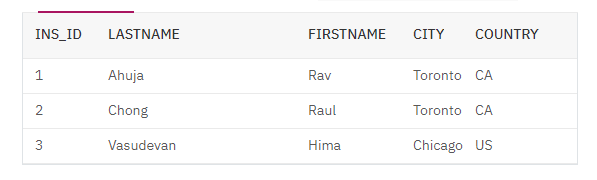
Task 2B: Insert two rows at once in the INSTRUCTOR table for instructors Raul Chong and Hima Vasudevan.

(Hint: list the values for the second row after the first row)

C:\Users\pares\Desktop\SQL(EDX)\17.png

Task 3: Select all rows from the INSTRUCTOR table.

C:\Users\pares\Desktop\SQL(EDX)\19.png



Task 3B: Select the firstname, lastname and country where the city is Toronto

C:\Users\pares\Desktop\SQL(EDX)\20.png

Task 5: Delete the row for Raul Chong from the table.

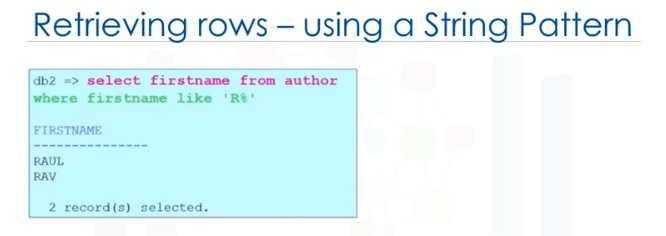
C:\Users\pares\Desktop\SQL(EDX)\22.png

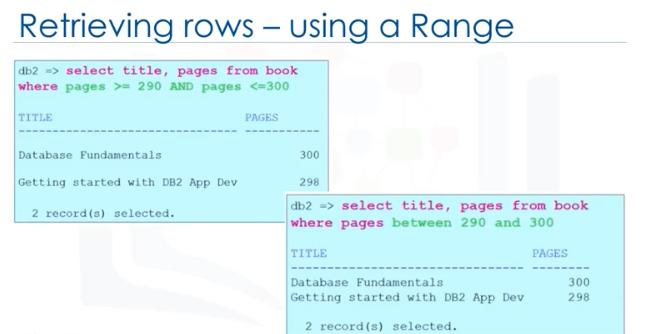
Task 5B: Retrieve all rows in the INSTRUCTOR table

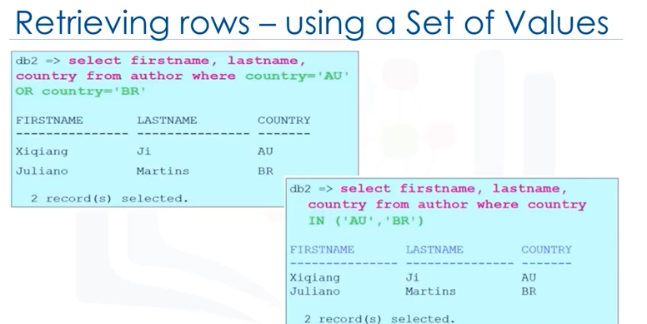
C:\Users\pares\Desktop\SQL(EDX)\23.png

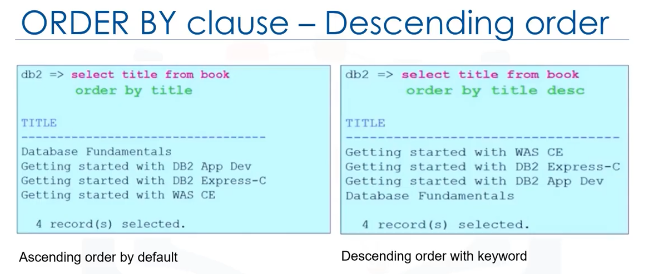
Module-3-

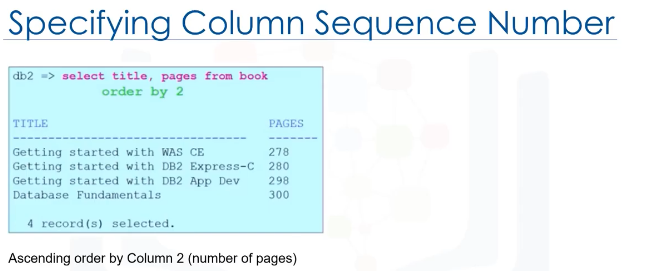
By the end of this module you will have learnt how to use string patterns and ranges to search data and how to sort and group data in result sets.

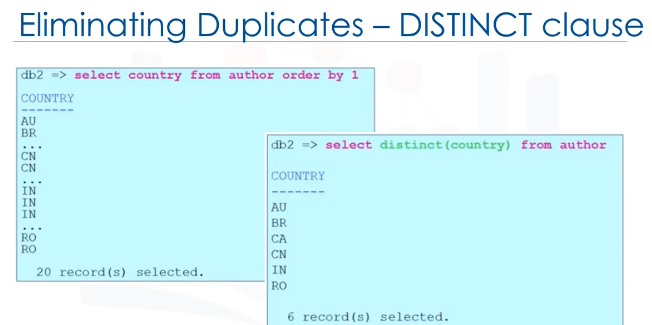




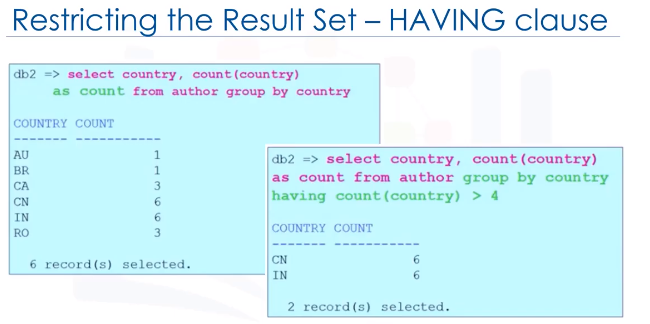












Refer Lab3 folder.

Note- WHERE clause is used for filtering the entire result set whereas the HAVING clause is used for filtering the result of the grouping.

-- Query 1------

;

select F\_NAME , L\_NAME

from EMPLOYEES

where ADDRESS LIKE '%Elgin,IL%' ;

--Query 2--

;

select F\_NAME , L\_NAME

from EMPLOYEES

where B\_DATE LIKE '197%' ;

---Query3--

;

select \*

from EMPLOYEES

where (SALARY BETWEEN 60000 and 70000) and DEP\_ID = 5 ;

--Query4A--

;

select F\_NAME, L\_NAME, DEP\_ID

from EMPLOYEES

order by DEP\_ID;

--Query4B--

;

select F\_NAME, L\_NAME, DEP\_ID

from EMPLOYEES

order by DEP\_ID desc, L\_NAME desc;

--Query5A--

;

select DEP\_ID, COUNT(\*)

from EMPLOYEES

group by DEP\_ID;

--Query5B--

;

select DEP\_ID, COUNT(\*), AVG(SALARY)

from EMPLOYEES

group by DEP\_ID;

--Query5C--

;

select DEP\_ID, COUNT(\*) AS "NUM\_EMPLOYEES", AVG(SALARY) AS "AVG\_SALARY"

from EMPLOYEES

group by DEP\_ID;

--Query5D--

;

select DEP\_ID, COUNT(\*) AS "NUM\_EMPLOYEES", AVG(SALARY) AS "AVG\_SALARY"

from EMPLOYEES

group by DEP\_ID

order by AVG\_SALARY;

--Query5E--

;

select DEP\_ID, COUNT(\*) AS "NUM\_EMPLOYEES", AVG(SALARY) AS "AVG\_SALARY"

from EMPLOYEES

group by DEP\_ID

having count(\*) < 4

order by AVG\_SALARY;

--5E alternative: if you want to use the label

select DEP\_ID, NUM\_EMPLOYEES, AVG\_SALARY from

( select DEP\_ID, COUNT(\*) AS NUM\_EMPLOYEES, AVG(SALARY) AS AVG\_SALARY from EMPLOYEES group by DEP\_ID)

where NUM\_EMPLOYEES < 4

order by AVG\_SALARY;

--BONUS Query6--

;

select D.DEP\_NAME , E.F\_NAME, E.L\_NAME

from EMPLOYEES as E, DEPARTMENTS as D

where E.DEP\_ID = D.DEPT\_ID\_DEP

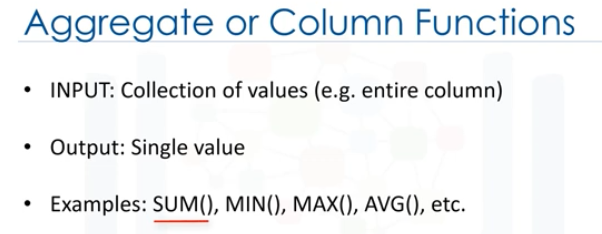
order by D.DEP\_NAME, E.L\_NAME desc ;

Module-4

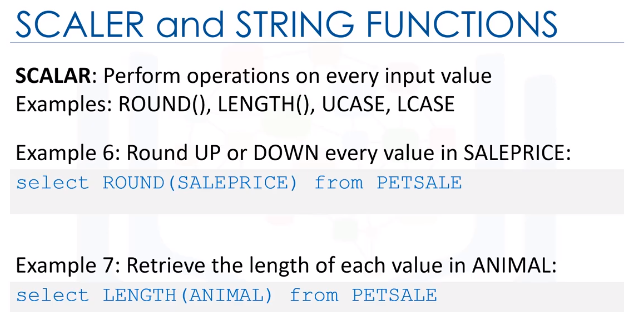
By the end of this module you will have learnt how to invoke built-in database functions, work with nested queries, and run involving multiple tables.

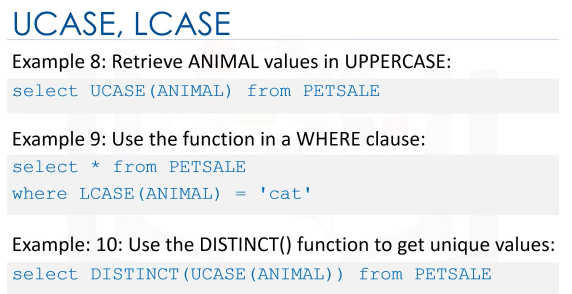
Types of Functions-

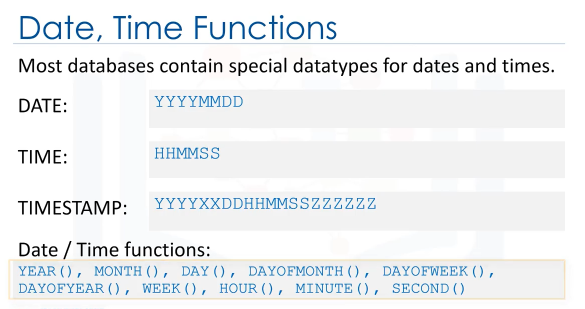
* Built in function
* User defined function

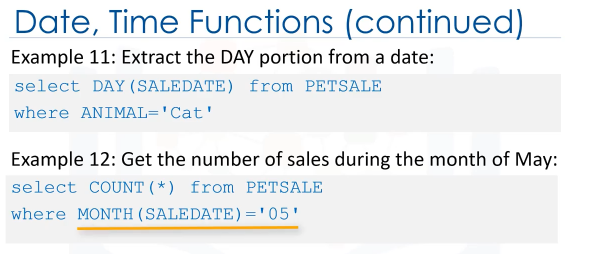


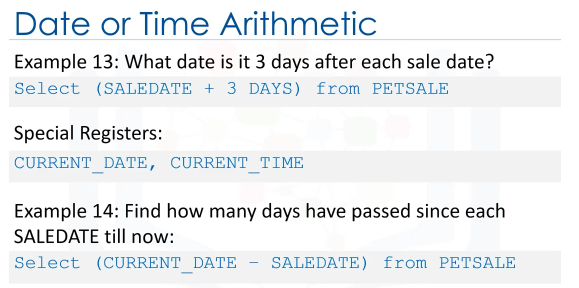


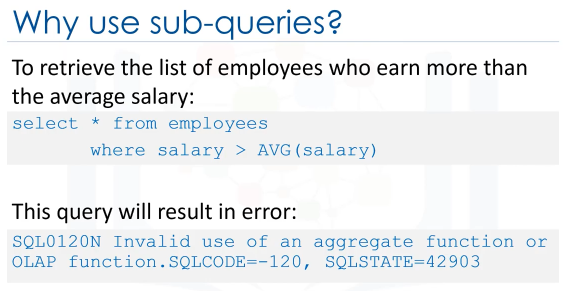


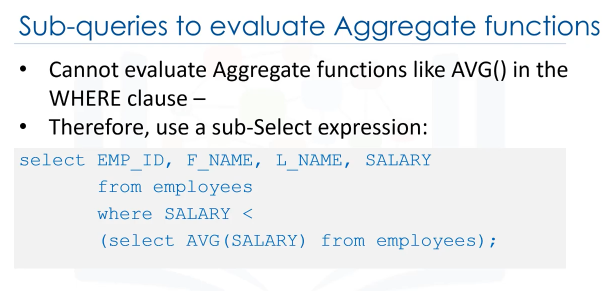


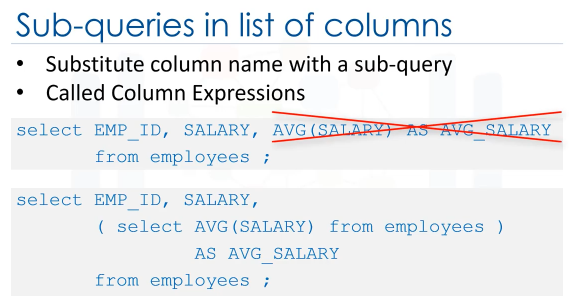


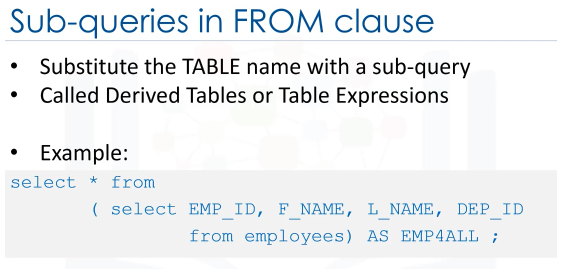


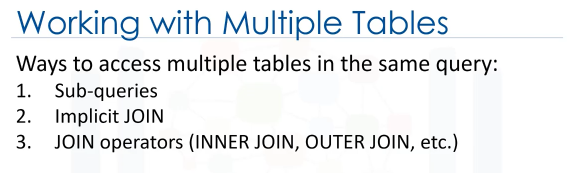


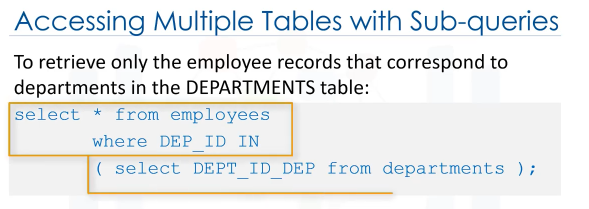


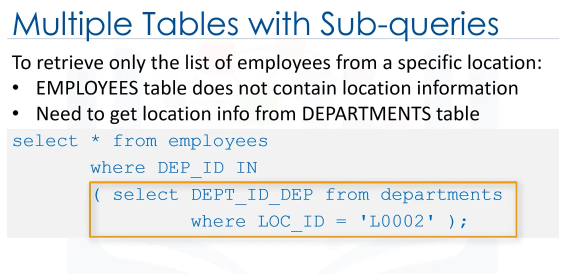


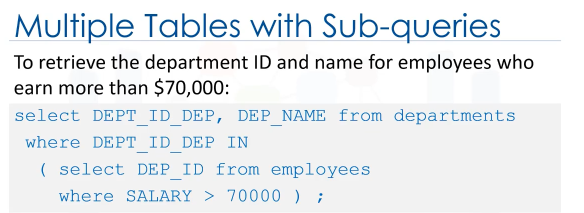




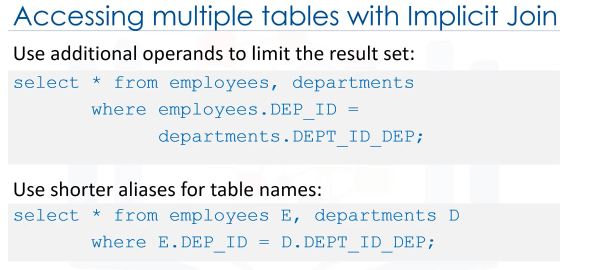


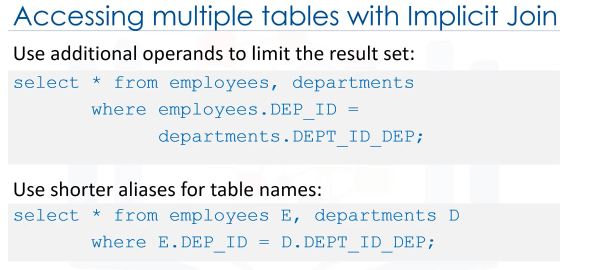


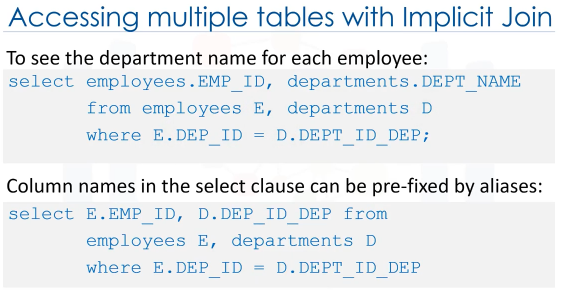












**LAB: Practice Sub-Queries and Working with Multiple Tables**

Now lets practice sub-queries and working with multiple tables. Use the EMPLOYEES and DEPARTMENTS tables created previously and execute the queries in the last two lessons.

Here are the queries to save you some time typing them. Please execute these queries and verify the results:

**Sub-Queries and Nested-Selects**

Query A1: Enter a failing (i.e. which gives an error) to retrieve all employees whose salary is greater than the average salary

select \* from employees where salary > AVG(salary)

Query A2: Enter a working query using a sub-select to retrieve all employees whose salary is greater than the average salary

select EMP\_ID, F\_NAME, L\_NAME, SALARY from employees where SALARY > (select AVG(SALARY) from employees);

Query A3: Enter a failing query (i.e.  that gives an error) that retrieves all employees records and average salary in every row

select EMP\_ID, SALARY, AVG(SALARY) AS AVG\_SALARY from employees ;

Query A4: Enter a Column Expression that retrieves all employees records and average salary in every row

select EMP\_ID, SALARY, ( select AVG(SALARY) from employees ) AS AVG\_SALARY from employees ;

Query A5: Enter a Table Expression that retrieves only the columns  with non-sensitive employee data

select \* from ( select EMP\_ID, F\_NAME, L\_NAME, DEP\_ID from employees) AS EMP4ALL ;

**Part B: Accessing Multiple Tables with Sub-Queries**

Query B1: Retrieve only the EMPLOYEES records that correspond to departments in the DEPARTMENTS table

select \* from employees where DEP\_ID IN ( select DEPT\_ID\_DEP from departments );

Query B2: Retrieve only the list of employees from location L0002

select \* from employees where DEP\_ID IN ( select DEPT\_ID\_DEP from departments where LOC\_ID = 'L0002' );

Query B3: Retrieve the department ID and name for employees who earn more than $70,000

select DEPT\_ID\_DEP, DEP\_NAME from departments where DEPT\_ID\_DEP IN ( select DEP\_ID from employees where SALARY > 70000 ) ;

Query B4: Specify 2 tables in the FROM clause

select \* from employees, departments;

**Accessing Multiple Tables with Implicit Joins**

Query B5: Retrieve only the EMPLOYEES records that correspond to departments in the DEPARTMENTS table

select \* from employees, departments where employees.DEP\_ID = departments.DEPT\_ID\_DEP;

Query B6: Use shorter aliases for table names

select \* from employees E, departments D where E.DEP\_ID = D.DEPT\_ID\_DEP;

Query B7: Retrieve only the Employee ID and Department name in the above query

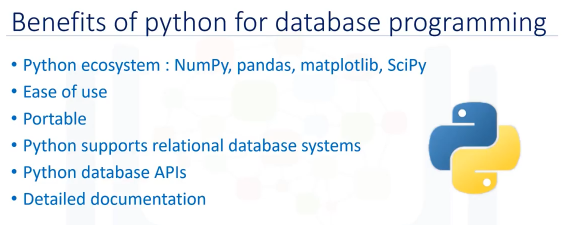
select EMP\_ID, DEP\_NAME from employees E, departments D where E.DEP\_ID = D.DEPT\_ID\_DEP;

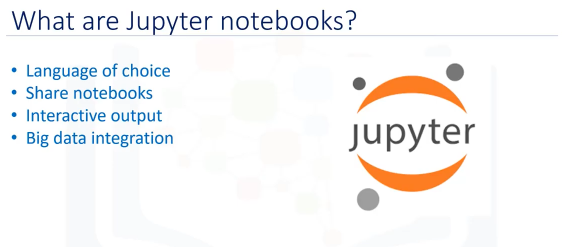
Query B8: In the above query specify the fully qualified column names with aliases in the SELECT clause

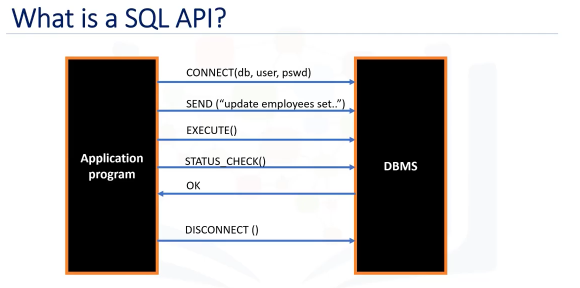
select E.EMP\_ID, D.DEP\_NAME from employees E, departments D where E.DEP\_ID = D.DEPT\_ID\_DEP

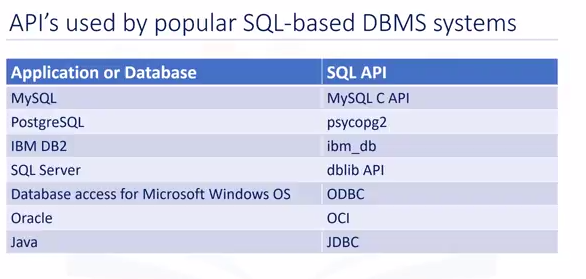
Module 5-

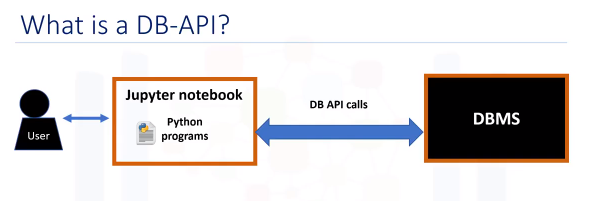
After completing the lessons in this week, you will know how to explain the basic concepts related to using Jupyter notebooks to connect to databases and then create tables, load data, query data using SQL and analyze data using Python.

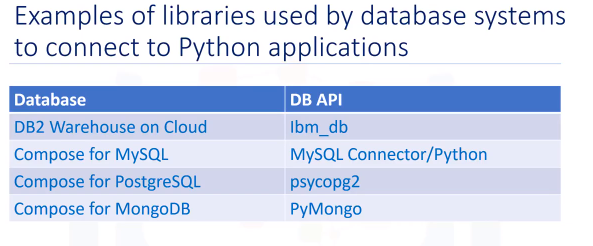








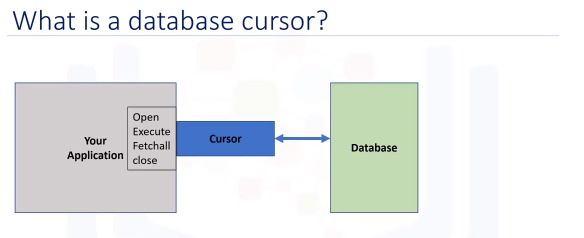


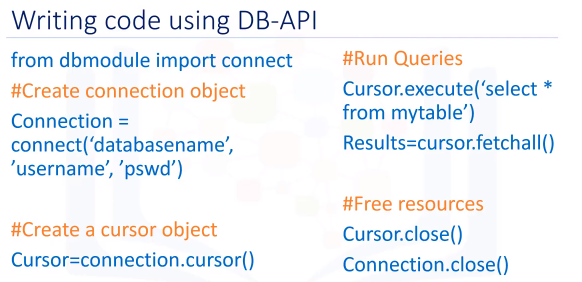


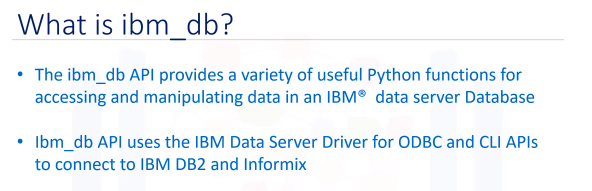


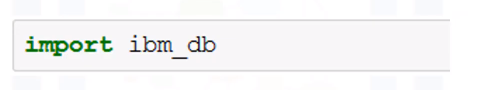






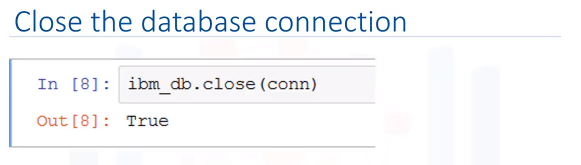












Go through pdf for Lab5 for connection details.

{

"db": "BLUDB",

"dsn": "DATABASE=BLUDB;HOSTNAME=dashdb-txn-sbox-yp-lon02-04.services.eu-gb.bluemix.net;PORT=50000;PROTOCOL=TCPIP;UID=rcm78049;PWD=mxtzzbnh@svr0rng;",

"host": "dashdb-txn-sbox-yp-lon02-04.services.eu-gb.bluemix.net",

"hostname": "dashdb-txn-sbox-yp-lon02-04.services.eu-gb.bluemix.net",

"https\_url": "https://dashdb-txn-sbox-yp-lon02-04.services.eu-gb.bluemix.net:8443",

"jdbcurl": "jdbc:db2://dashdb-txn-sbox-yp-lon02-04.services.eu-gb.bluemix.net:50000/BLUDB",

"parameters": {},

"password": "mxtzzbnh@svr0rng",

"port": 50000,

"ssldsn": "DATABASE=BLUDB;HOSTNAME=dashdb-txn-sbox-yp-lon02-04.services.eu-gb.bluemix.net;PORT=50001;PROTOCOL=TCPIP;UID=rcm78049;PWD=mxtzzbnh@svr0rng;Security=SSL;",

"ssljdbcurl": "jdbc:db2://dashdb-txn-sbox-yp-lon02-04.services.eu-gb.bluemix.net:50001/BLUDB:sslConnection=true;",

"uri": "db2://rcm78049:mxtzzbnh%40svr0rng@dashdb-txn-sbox-yp-lon02-04.services.eu-gb.bluemix.net:50000/BLUDB",

"username": "rcm78049"

}

This notebook illustrates how to access a DB2 database on Cloud using Python by following the steps below:

1. Import the ibm\_db Python library
2. Enter the database connection credentials
3. Create the database connection
4. Close the database connection

Here we established a connection to a DB2 database on Cloud database from a Python notebook using ibm\_db API.

Step 1-

Import sql module-

import ibm\_db

Step 2-

Identify database connection credentials

#Replace the placeholder values with your actual Db2 hostname, username, and password:

dsn\_hostname = "dashdb-txn-sbox-yp-lon02-04.services.eu-gb.bluemix.net" # e.g.: "dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net"

dsn\_uid = "rcm78049" # e.g. "abc12345"

dsn\_pwd = "mxtzzbnh@svr0rng" # 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"

Step 3-

Create Db2 database connection

#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)

#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() )

#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)

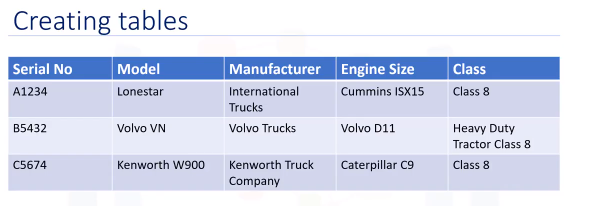
print ("APPL\_CODEPAGE: ", client.APPL\_CODEPAGE)

print ("CONN\_CODEPAGE: ", client.CONN\_CODEPAGE)

Step 4-

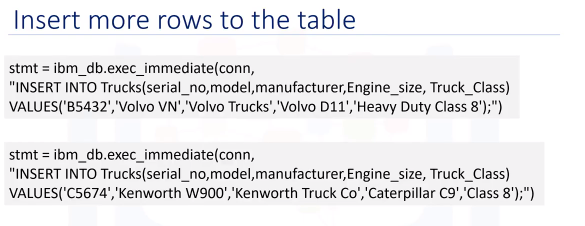
Close connection

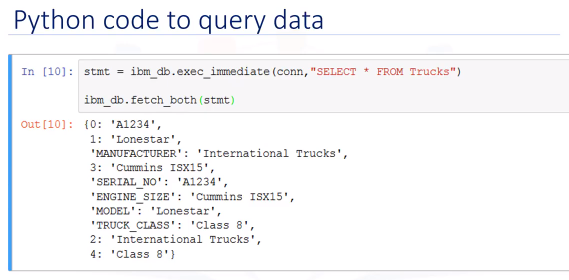
ibm\_db.close(conn)

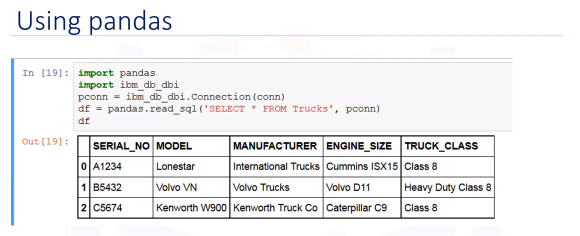










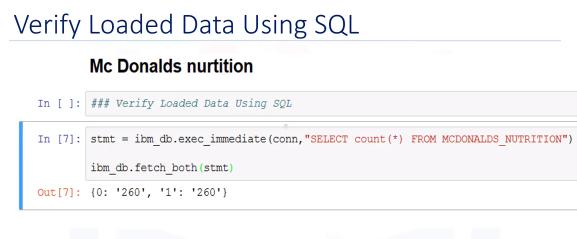


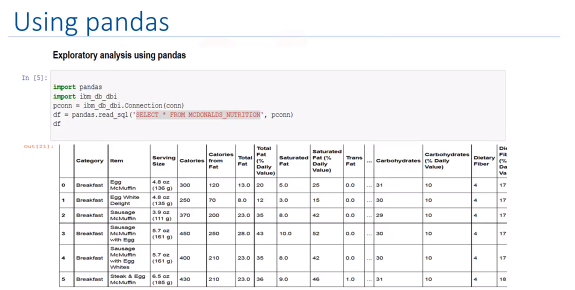
Refere DB0201EN-Week3-1-2-Querying-v4-py in Lab5 for creating table in sql using python.

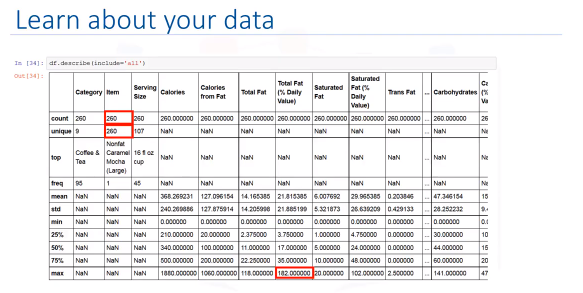
<https://labs.cognitiveclass.ai/tools/jupyterlab/lab/tree/labs/DB0201EN/DB0201EN-Week3-1-4-Analyzing-v5-py.ipynb>

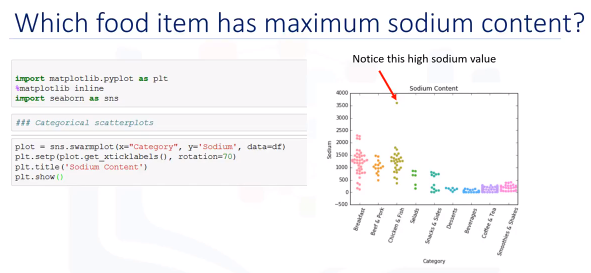


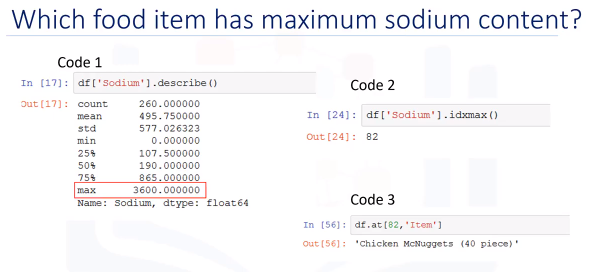




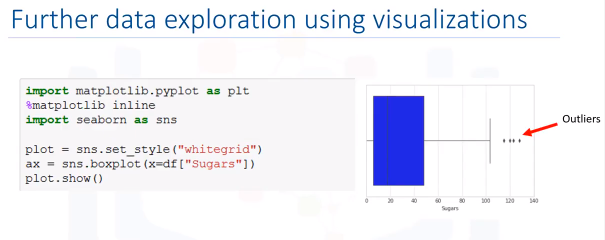












Module 6-

As a hands-on Data Scientist, you will be working with multiple real world datasets. In this module we review working with real world data sets and then perform an assignment utilizing a real-world data set. As part of the assignment you will be asked questions that will help you understand the data just like a data scientist would. You will be assessed both on the correctness of your SQL queries and results.

