Contents

Introduction

Introduction to Databases

Course Overview

- Basics of SQL
- Relational Database Model

- At the end of this course, you will be able to discuss SQL basics and explain aspects of the relational database model

- At the end of this lesson, you will be able to:
  - Describe SQL, data, database, relational database
  - List five basic SQL commands

What is SQL?

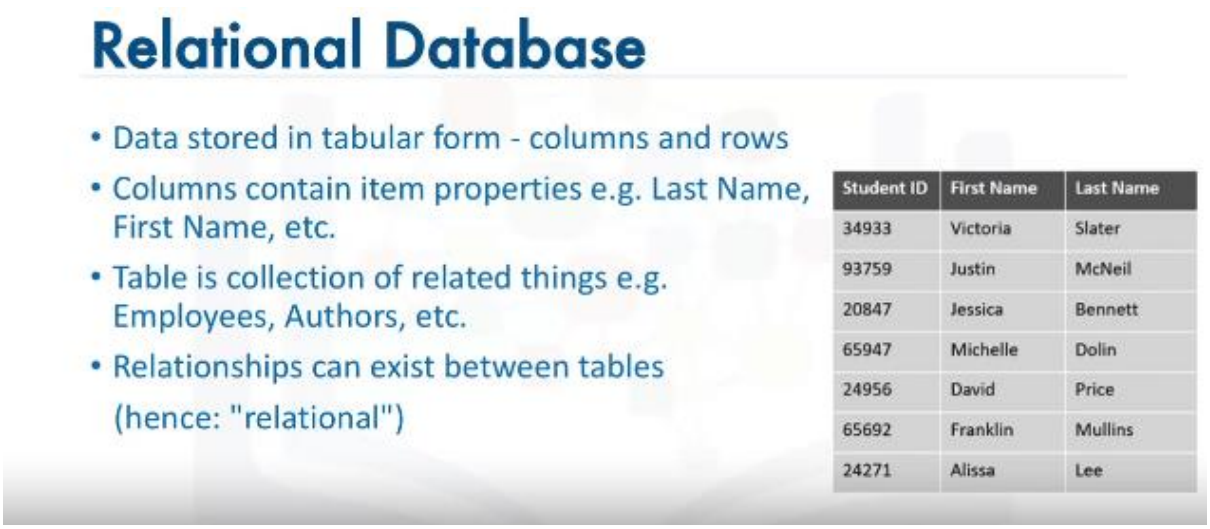
- A language used for relational databases
- Query data

What is data?

- Facts (words, numbers)
- Pictures
- One of the most critical assets of any business
- Needs to be secure

What 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



DBMS & RDBMS

Basic SQL Commands

Basic SQL Commands

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

SELECT Statement

At the end of this video, you will be able to:

- Retrieve data from a relational database table
- Define the use of a predicate
- Identify the syntax of the SELECT statement using the WHERE clause
- List the comparison operators supported by a RDBMS

Retrieving rows from a table

- After creating a table and inserting data into the table, we want to see the data
- SELECT statement
  - A Data Manipulation Language (DML) statement used to read and modify data

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

Select * from <tablename>
```

Using the SELECT Statement

Title	Edition	Year	Price	ISBN	Pages	Aisle	Description
Database Fundamentals	1	2010	24.99	978-0-9800628 3-1-1	300	DB-A02	Teaches you the fundamentals of databases
Getting started with DB2 Express-C	1	2010	24.99	978-0-9866628 3-5-1	280	DB-A01	Teaches you the essentials of DB2 using DB2 Express-C

Example: select * from Book

```
db2 => select * from Book
```

Book_ID	Title	Edition	Year	Price	ISBN	Pages	Aisle	Description
B1	Getting started with DB2 Express-C	1	2010	24.99	978-0-98666283-5-1	280	DB-A01	Teaches you the essentials of DB2 using DB2 Express-C
B2	Database Fundamentals	1	2010	24.99	978-0-98006283-1-1	300	DB-A02	Teaches you the fundamentals of databases
B3	Getting started with DB2 App Dev	1	2011	35.99	978-0-98086283-4-1	345	DB-A03	Teaches you the essentials of developing applications for DB2.
B4	Getting started with WAS CE	1	2010	49.99	978-0-98946283-3-1	458	DB-A04	Teaches you the essentials of WebSphere Application Server

4 record(s) selected.

Using the SELECT Statement

Title	Edition	Year	Price	ISBN	Pages	Aisle	Description
Database Fundamentals	1	2010	24.99	978-0-9800628 3-1-1	300	DB-A02	Teaches you the fundamentals of databases
Getting started with DB2 Express-C	1	2010	24.99	978-0-9866628 3-5-1	280	DB-A01	Teaches you the essentials of DB2 using DB2 Express-C

Example: select <column 1, column 2, ..., column n from Book

```
db2 => select book_id, title, edition, year, price, ISBN, pages, aisle, description from Book
```

Book_ID	Title	Edition	Year	Price	ISBN	Pages	Aisle	Description
B1	Getting started with DB2 Express-C	1	2010	24.99	978-0-98666283-5-1	280	DB-A01	Teaches you the essentials of DB2 using DB2 Express-C
B2	Database Fundamentals	1	2010	24.99	978-0-98006283-1-1	300	DB-A02	Teaches you the fundamentals of databases
B3	Getting started with DB2 App Dev	1	2011	35.99	978-0-98086283-4-1	345	DB-A03	Teaches you the essentials of developing applications for DB2.
B4	Getting started with WAS CE	1	2010	49.99	978-0-98946283-3-1	458	DB-A04	Teaches you the essentials of WebSphere Application Server

4 record(s) selected.

Retrieving a subset of the columns

- You can retrieve just the columns you want
- The order of the columns displayed always matches the order in the SELECT statement
  - SELECT <column 1>, <column 2> from Book

```
db2 => select book_id, title from Book
```

Book_ID	Title
B1	Getting started with DB2 Express-C
B2	Database Fundamentals
B3	Getting started with DB2 App Dev
B4	Getting started with WAS CE

4 record(s) selected.

Restricting the Result Set: WHERE Clause

- Restricts the result set
- Always requires a Predicate:
  - Evaluates to:
    True, False or Unknown
  - Used in the search condition of the Where clause

```
select book_id, title from Book
       WHERE predicate
```

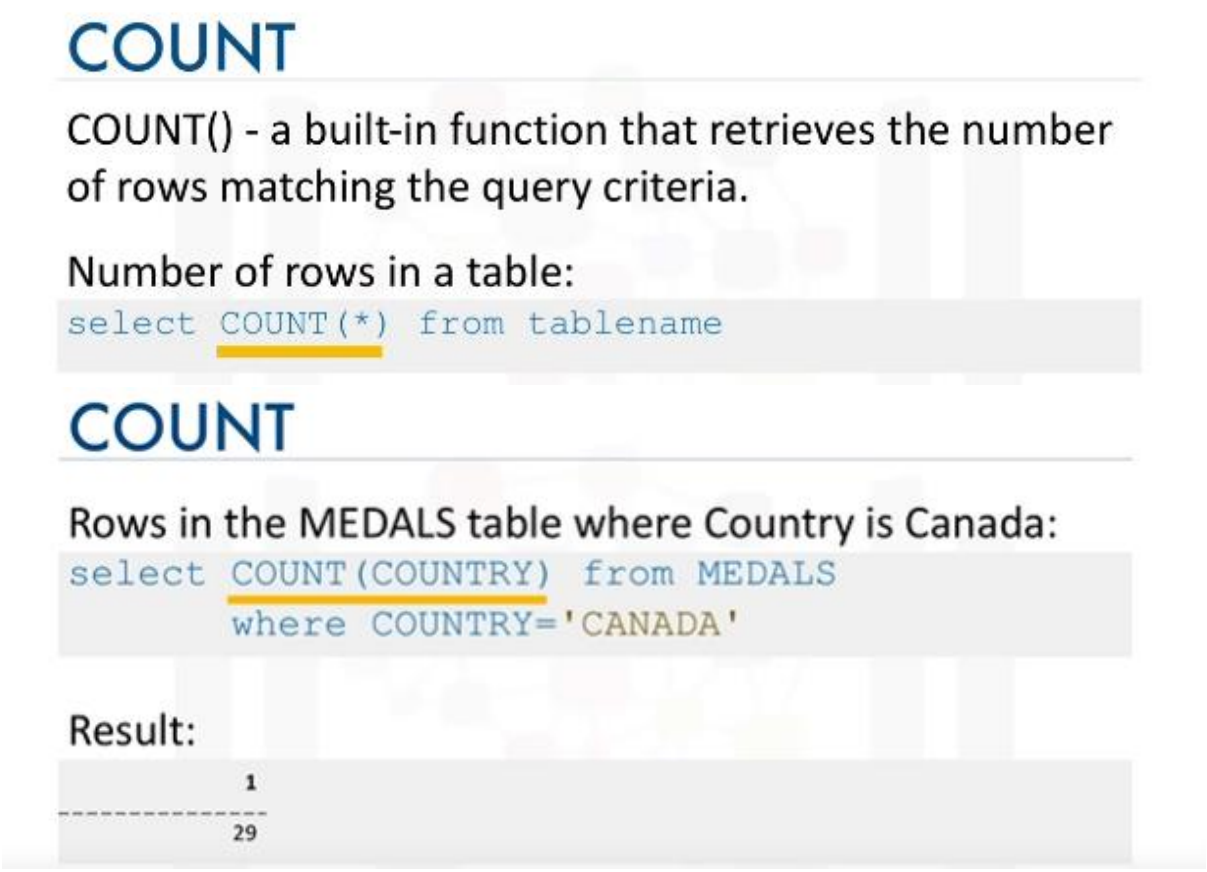
```
db2 => select book_id, title from Book
       WHERE book_id='B1'

Book_ID    Title
-----------------------------------------------------
B1         Getting started with DB2 Express-C

  1 record(s) selected
```

WHERE Clause Comparison Operators

```
select book_id, title from Book
     WHERE book_id = 'B1'
```

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

COUNT


COUNT
COUNT() - a built-in function that retrieves the number of rows matching the query criteria.
Number of rows in a table:
select COUNT(*) from tablename
COUNT
Rows in the MEDALS table where Country is Canada:
select COUNT(COUNTRY) from MEDALS
where COUNTRY='CANADA'
Result:
1
29

DISTINCT


DISTINCT
DISTINCT is used to remove duplicate values from a result set.
Retrieve unique values in a column:
select DISTINCT columnname from tablename
DISTINCT
List of unique countries that received GOLD medals:
select DISTINCT COUNTRY from MEDALS
where MEDALTYPE = 'GOLD'
Result:
1
21

LIMIT

LIMIT

LIMIT is used for restricting the number of rows retrieved from the database.

Retrieve just the first 10 rows in a table:

```
select * from tablename LIMIT 10
```

LIMIT

Retrieve 5 rows in the MEDALS table for a particular year:

```
select * from MEDALS
       where YEAR = 2018 LIMIT 5
```

Result:

COUNTRY	GOLD	SILVER	BRONZE	TOTAL	YEAR
Norway	14	14	11	39	2018
Germany	14	10	7	31	2018
Canada	11	8	10	29	2018
United States	9	8	6	23	2018
Netherlands	8	6	6	20	2018

INSERT Statement

INSERT Statement

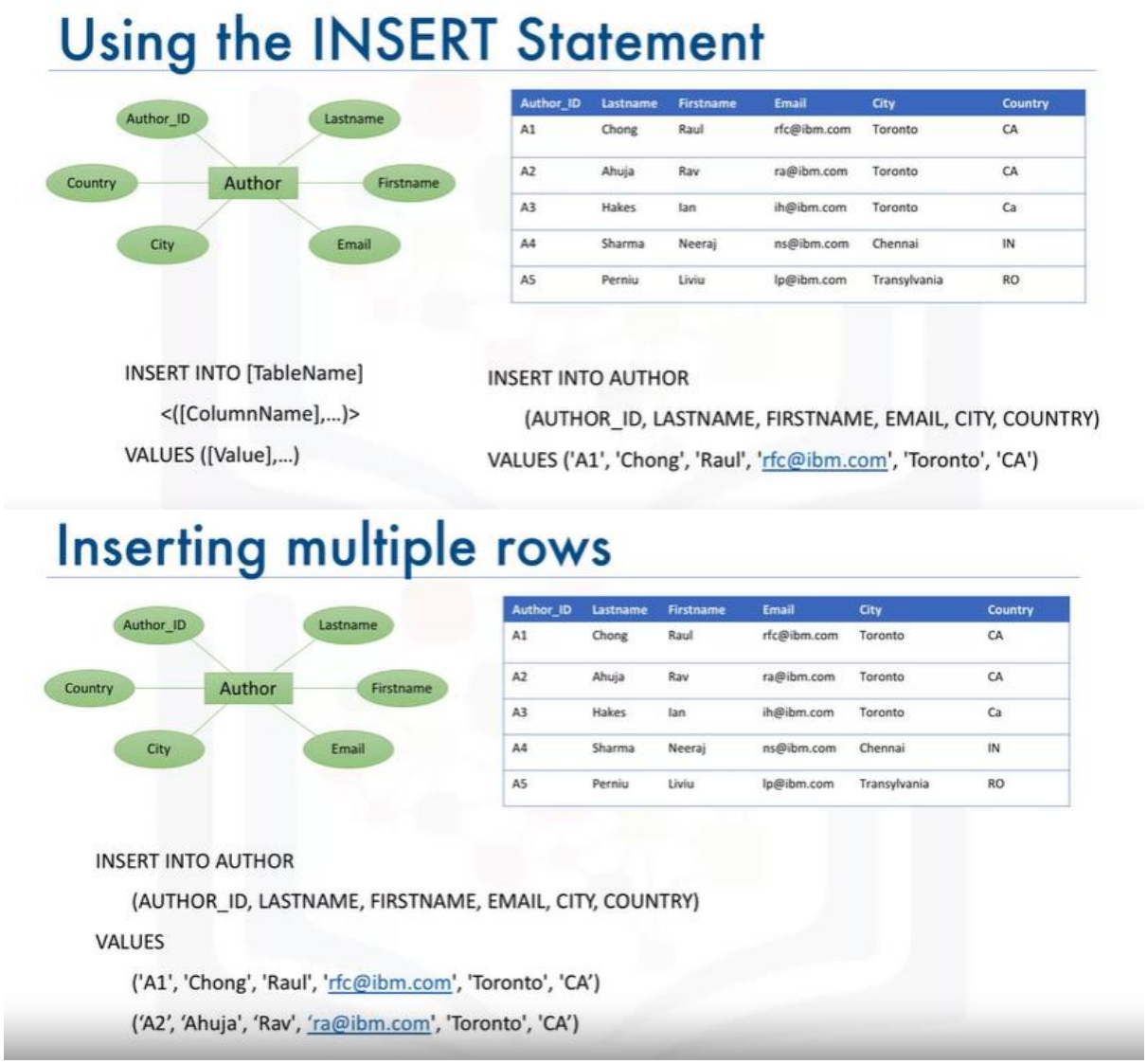
At the end of this video, you will be able to:

- Identify the syntax of the INSERT statement
- Explain two methods to add rows to a table

Adding rows to a table

- Create the table (CREATE TABLE statement)
- Populate table with data:
  - INSERT statement
    - a Data Manipulation Language (DML) statement used to read and modify data

Author_ID	Lastname	Firstname	Email	City	Country
A1	Chong	Raul	rfc@ibm.com	Toronto	CA
A2	Ahuja	Rav	ra@ibm.com	Toronto	CA
A3	Hakes	Ian	ih@ibm.com	Toronto	Ca
A4	Sharma	Neeraj	ns@ibm.com	Chennai	IN
A5	Perniu	Liviu	lp@ibm.com	Transylvania	RO



UPDATE & DELETE Statements

UPDATE & DELETE Statements

At the end of this lesson, you will be able to:

- Identify the syntax of the UPDATE statement
- Identify the syntax of the DELETE statement
- Explain the importance of the WHERE clause in both the UPDATE and DELETE statements

Altering rows of a table – UPDATE statement

- After creating a table and inserting data into the table, we can alter the data
  - UPDATE statement: A Data Manipulation Language (DML) statement used to read and modify data

Author_Id	LastName	FirstName	Email	City	Country
A1	Chong	Raul	rfc@ibm.com	Toronto	CA
A2	Ahuja	Rav	ra@ibm.com	Toronto	CA
A3	Hakes	Ian	ih@ibm.com	Toronto	CA
A4	Sharma	Neeraj	ns@ibm.com	Chennai	IN
A5	Perniu	Liviu	lp@ibm.com	Transylvania	RO

Using the UPDATE Statement

Author_Id	LastName	FirstName	Email	City	Country
A1	Chong	Raul	rfc@ibm.com	Toronto	CA
A2	Ahuja	Rav	ra@ibm.com	Toronto	CA
A3	Hakes	Ian	ih@ibm.com	Toronto	CA
A4	Sharma	Neeraj	ns@ibm.com	Chennai	IN
A5	Perniu	Liviu	lp@ibm.com	Transylvania	RO

```
UPDATE [TableName]
SET [[ColumnName]=[Value]]
<WHERE [Condition]>
```

Using the UPDATE Statement

Author_Id	LastName	FirstName	Email	City	Country
A1	CHONG	RAUL	rfc@ibm.com	Toronto	CA
A2	AHUJA	RAV	ra@ibm.com	Toronto	CA
A3	HAKES	IAN	ih@ibm.com	Toronto	CA

```
UPDATE AUTHOR
SET LASTNAME='KATTA'
    FIRSTNAME='LAKSHMI'
    WHERE AUTHOR_ID='A2'
```

Author_Id	LastName	FirstName	Email	City	Country
A1	CHONG	RAUL	rfc@ibm.com	Toronto	CA
A2	KATTA	LAKSHMI	ra@ibm.com	Toronto	CA
A3	HAKES	IAN	ih@ibm.com	Toronto	CA

Using the DELETE Statement

Author_Id	LastName	FirstName	Email	City	Country
A1	Chong	Raul	rfc@ibm.com	Toronto	CA
A2	Ahuja	Rav	ra@ibm.com	Toronto	CA
A3	Hakes	Ian	ih@ibm.com	Toronto	CA
A4	Sharma	Neeraj	ns@ibm.com	Chennai	IN
A5	Perniu	Liviu	lp@ibm.com	Transylvania	RO

```
DELETE FROM AUTHOR
   WHERE AUTHOR_ID IN ('A2', 'A3')
```

Author_Id	LastName	FirstName	Email	City	Country
A1	Chong	Raul	rfc@ibm.com	Toronto	CA
A4	Sharma	Neeraj	ns@ibm.com	Chennai	IN
A5	Perniu	Liviu	lp@ibm.com	Transylvania	RO

How does the syntax of an INSERT statement look?

```
INSERT INTO table_name (column1, column2, ... )
VALUES (value1, value2, ... )
;
```

How does the syntax of an UPDATE statement look?

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
```

```
WHERE condition;
```

How does the syntax of a DELETE statement look?

```
DELETE FROM table_name

WHERE condition

;
```

1.
    1. Problem:

       Update the city for Sandip to Toronto.

    2. Solution:

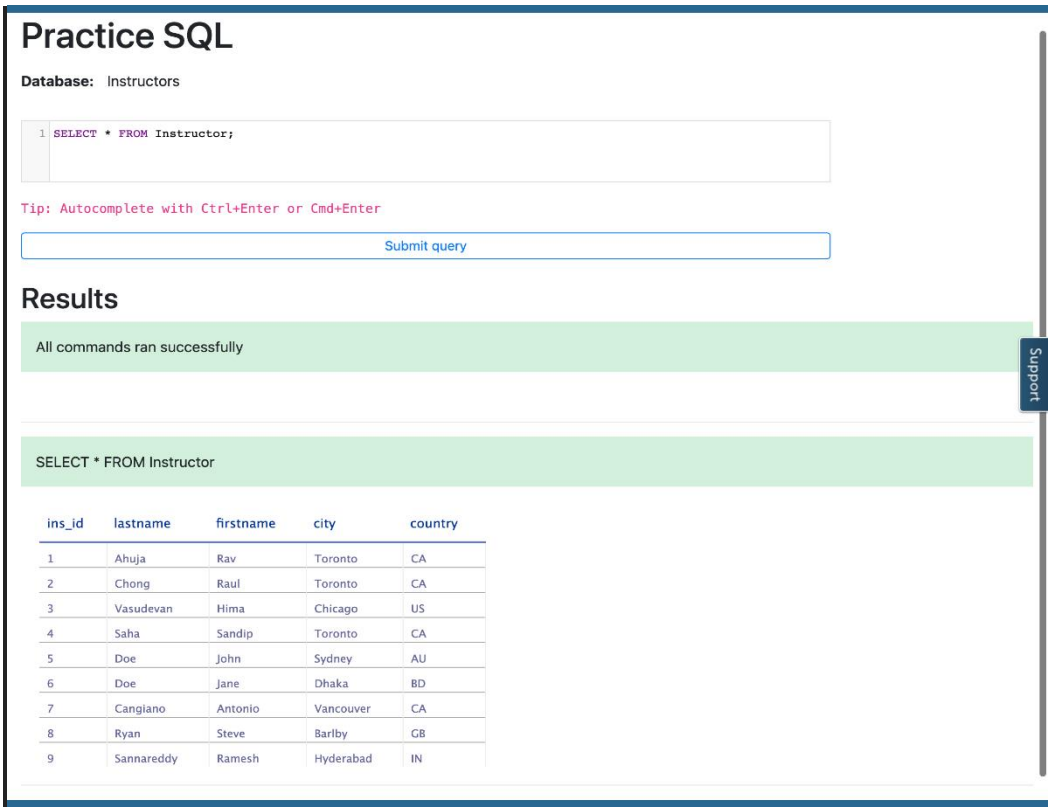
```
3. UPDATE Instructor
4. SET city='Toronto'
5. WHERE firstname="Sandip";
```

    6. Copy the solution code above by clicking on the little copy button on the bottom right of the codeblock below and paste it to the textbox of the Datasette tool. Then click **Submit query**.

    7. Copy the code below by clicking on the little copy button on the bottom right of the codeblock below and paste it to the textbox of the Datasette tool. Then click **Submit query**.

```
8. SELECT * FROM Instructor;
```

    9. Your output resultset should look like the image below:



2. In this example, we want to update multiple columns of an existing row of the table.
    1. Problem:

        Update the city and country for Doe with id 5 to Dubai and AE respectively.

    2. Solution:

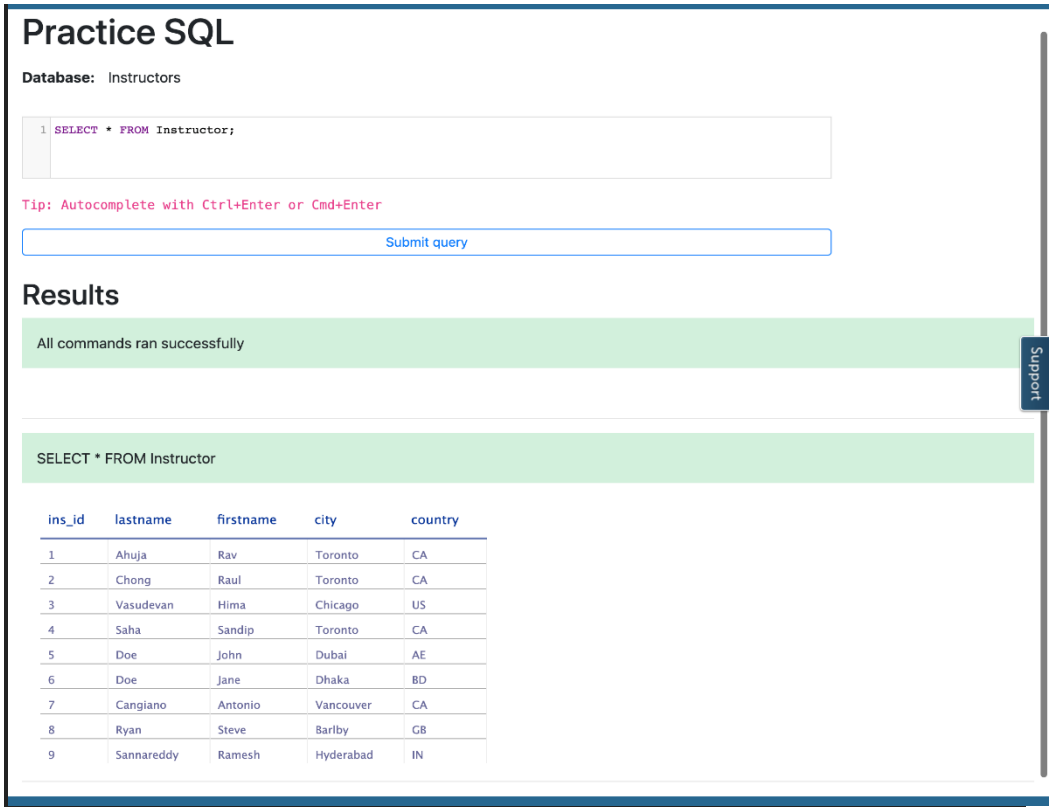
```
UPDATE Instructor
SET city='Dubai', country='AE'
WHERE ins_id=5;
```

6. Copy the solution code above by clicking on the little copy button on the bottom right of the codeblock below and paste it to the textbox of the Datasette tool. Then click **Submit query**.

7. Copy the code below by clicking on the little copy button on the bottom right of the codeblock below and paste it to the textbox of the Datasette tool. Then click **Submit query**.

```
SELECT * FROM Instructor;
```

9. Your output resultset should look like the image below:



In this example, we want to remove a row from the table.

1. Problem:

   Remove the instructor record of Doe whose id is 6.
2. Solution:

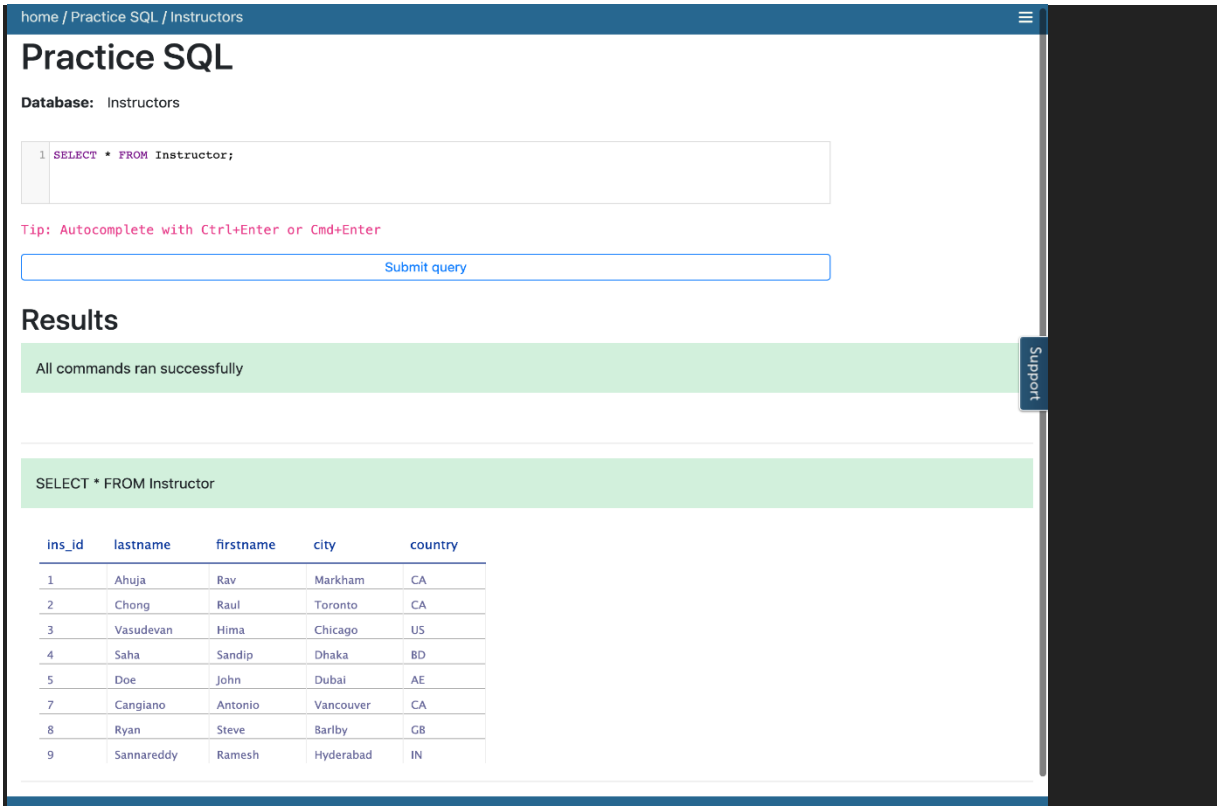
```
3. DELETE FROM instructor
4. WHERE ins_id = 6;
```

5. Copy the solution code above by clicking on the little copy button on the bottom right of the codeblock below and paste it to the textbox of **Custom SQL query** of the Datasette tool. Then click **Submit query**.

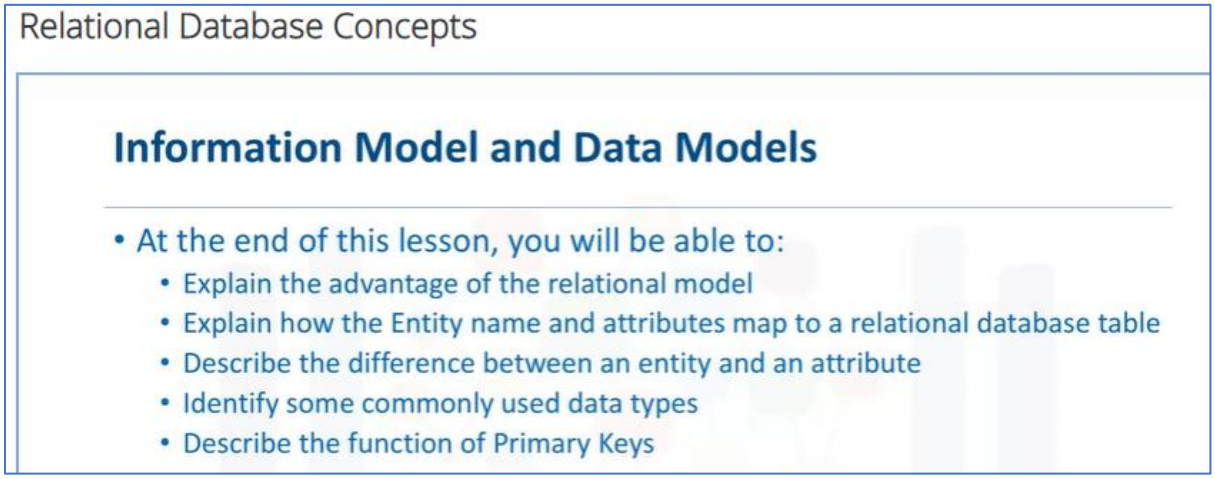
6. Copy the code below by clicking on the little copy button on the bottom right of the codeblock below and paste it to the textbox of the Datasette tool. Then click **Submit query**.

```
7. SELECT * FROM Instructor;
```

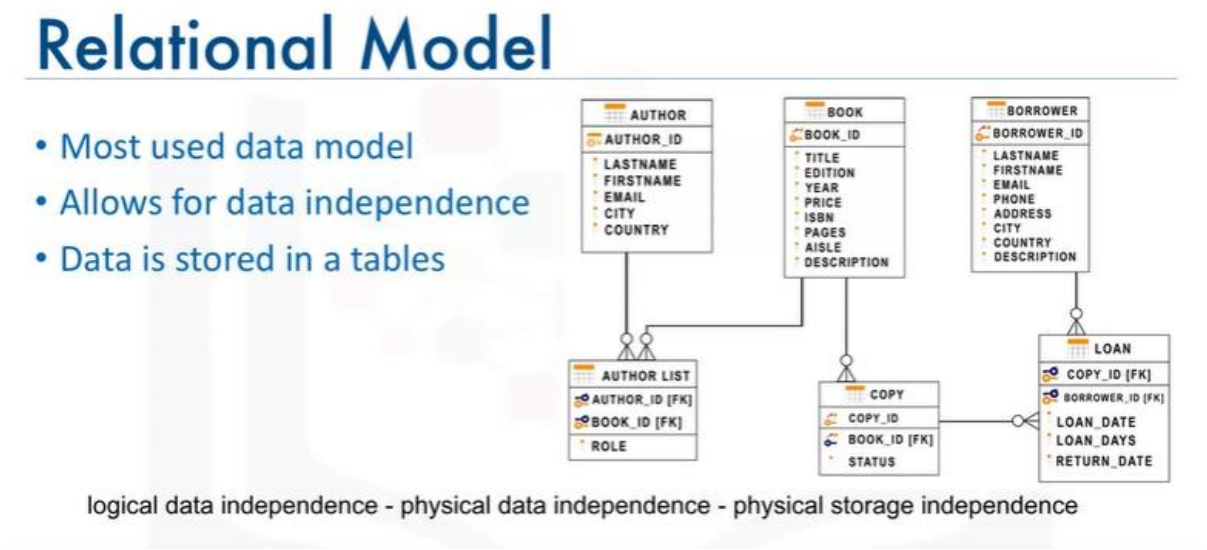
8. Your output resultset should look like the image below:



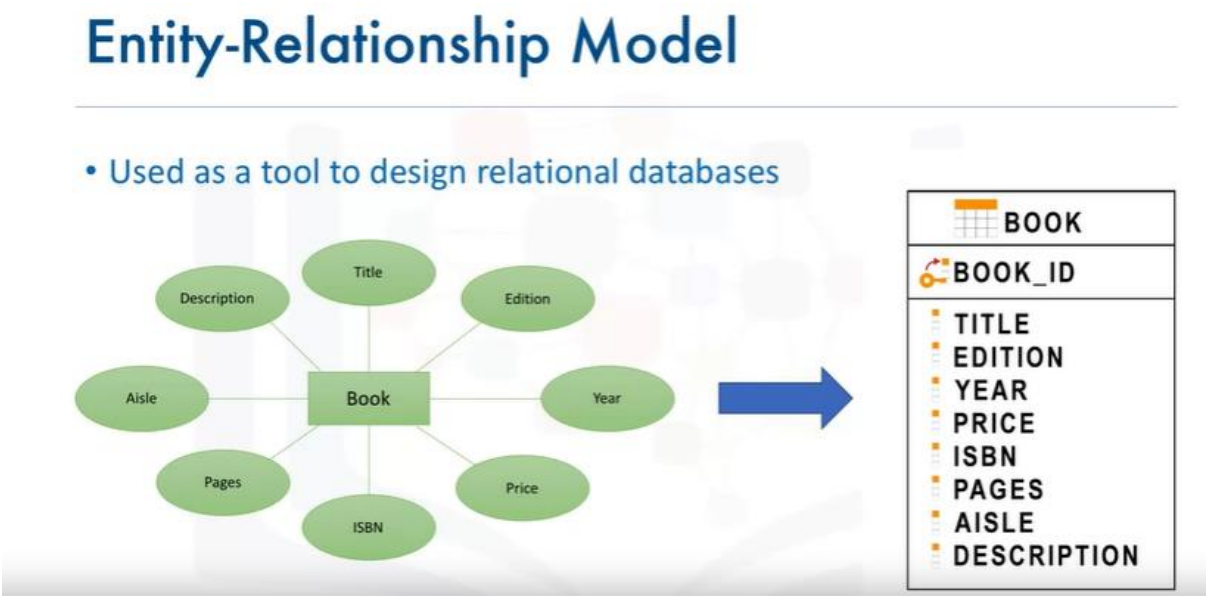
Relational Database Concepts



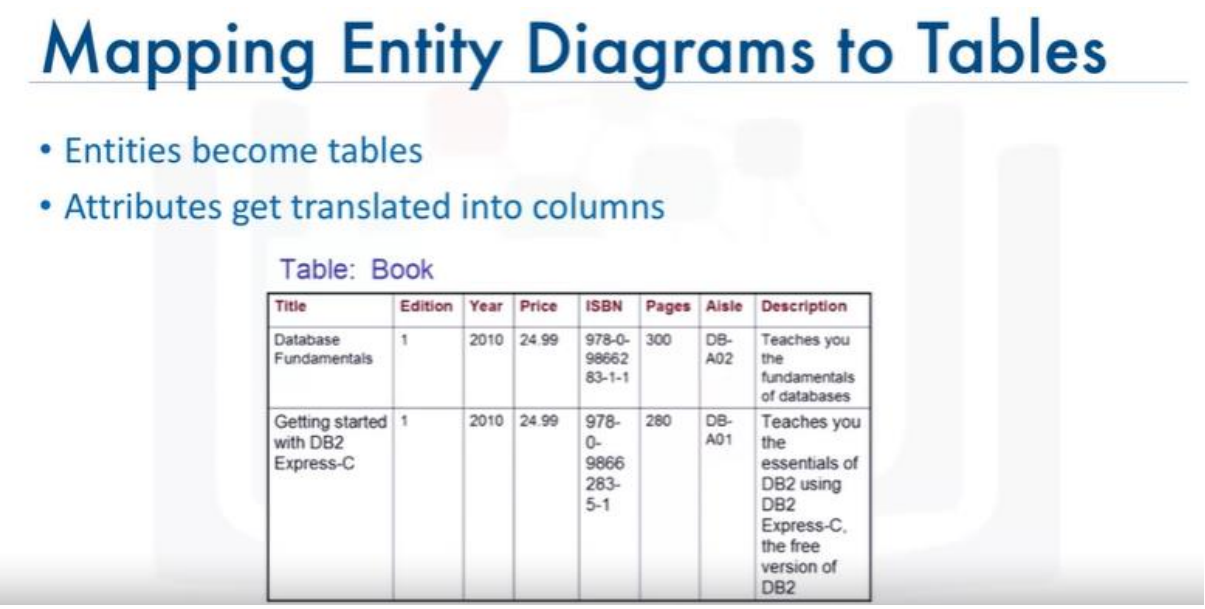
Relational Model



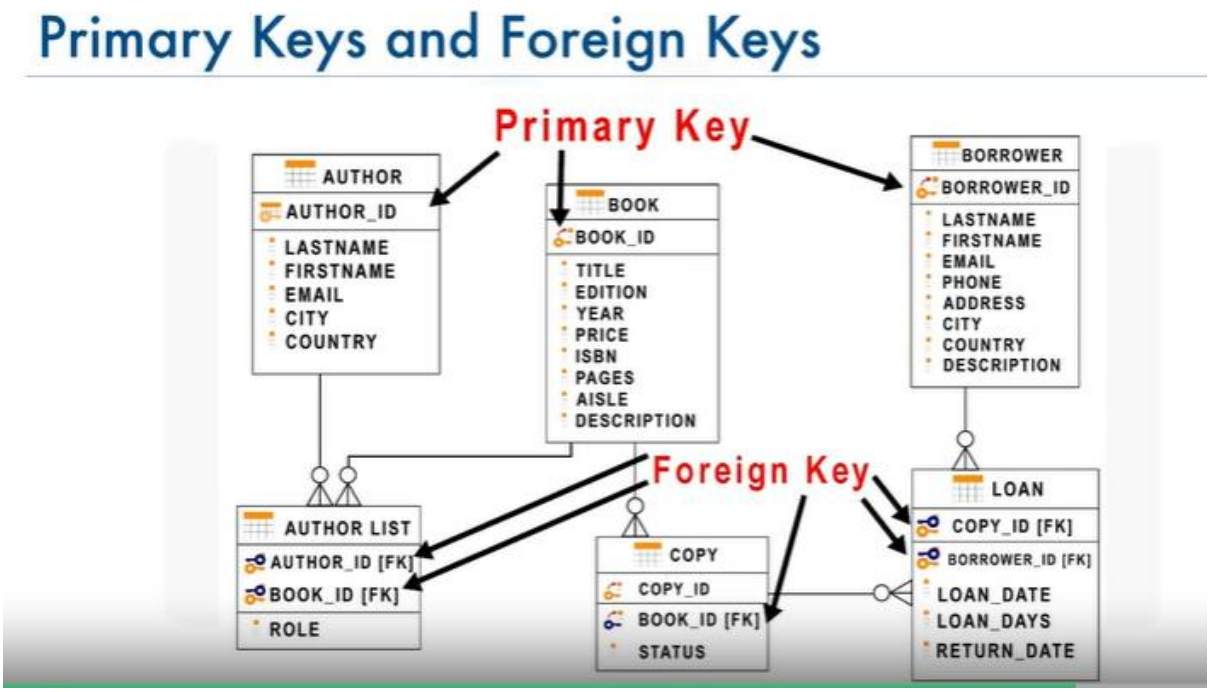
Entity-Relationship Model

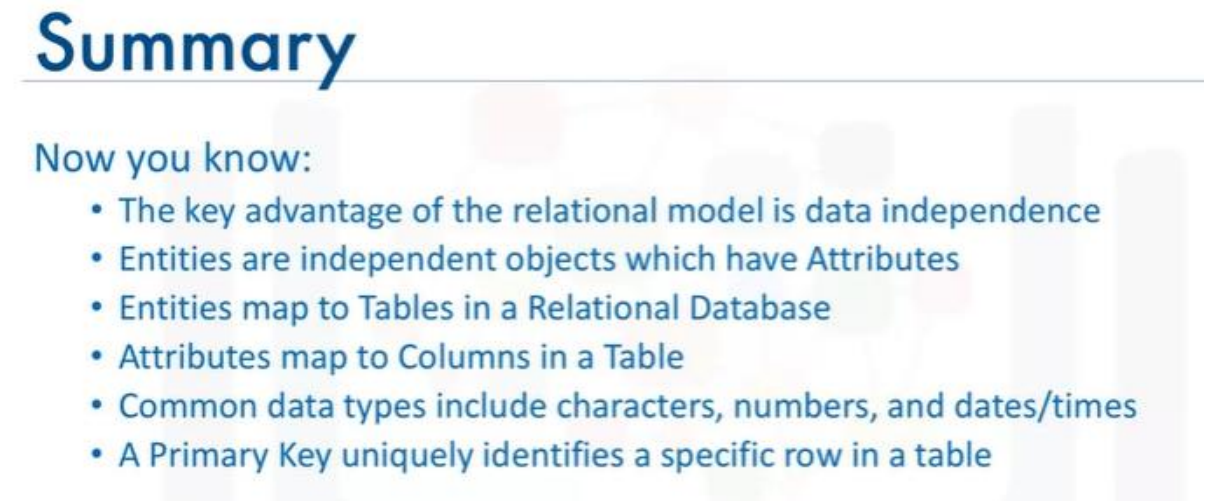


Mapping Entity Diagrams to Tables

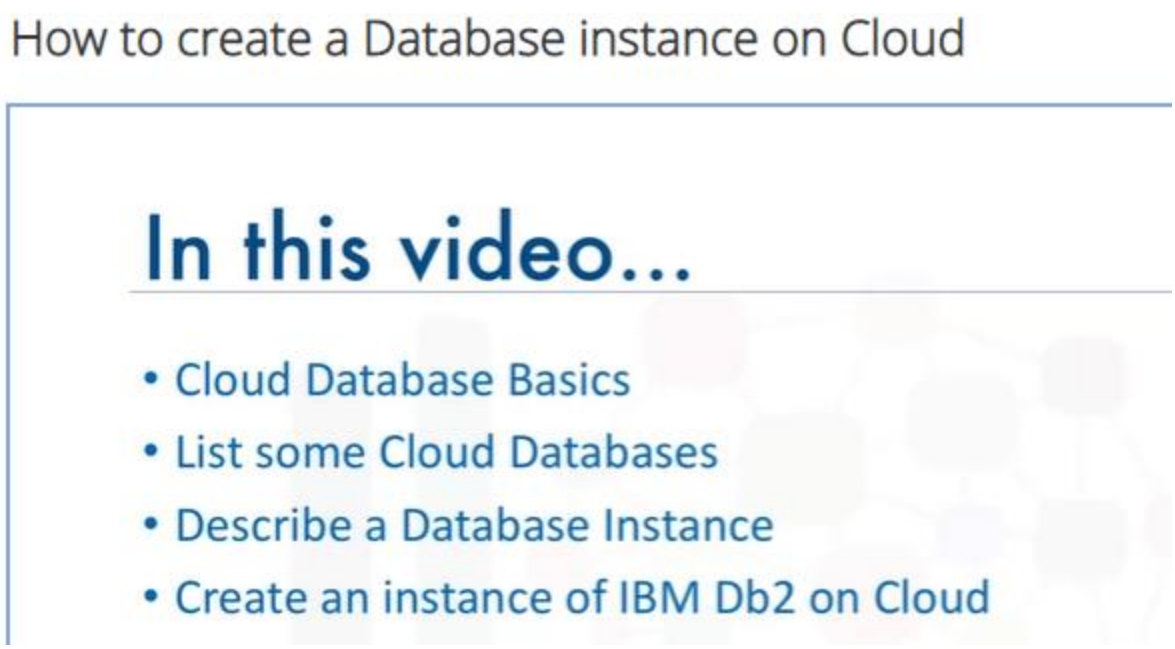


Primary Keys and Foregin Keys





How to craete a Database instance on Cloud



Cloud databases

Cloud databases

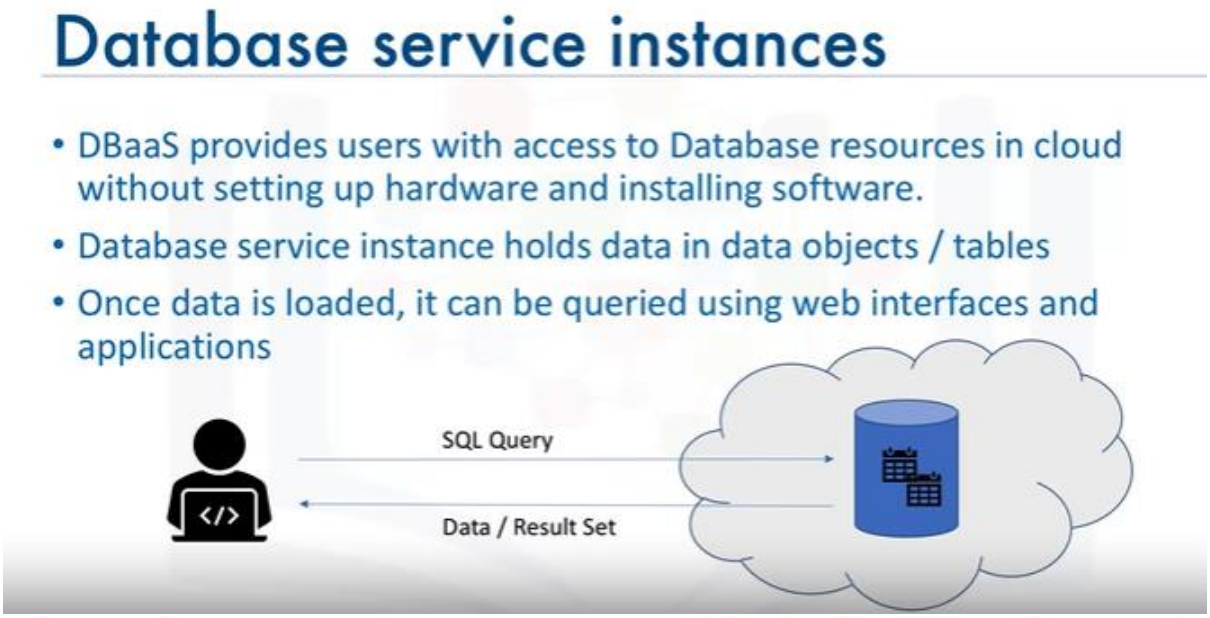
- ✓ Ease of Use and Access
  - API
  - Web Interface
  - Cloud or Remote Applications
- ✓ Scalability & Economics
  - Expand/Shrink Storage & Compute Resources
  - Pay per use
- ✓ Disaster Recovery
  - Cloud Backups and Geographical Distribution

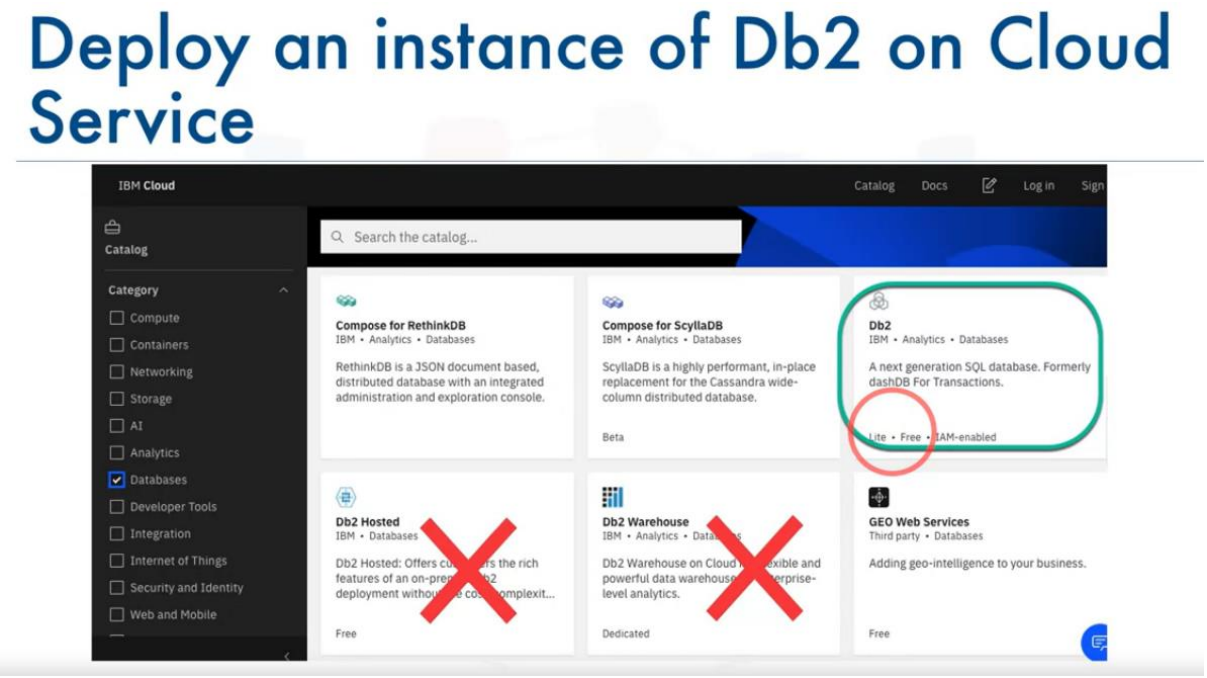
Examples of Cloud databases

- IBM Db2
- Databases for PostgreSQL
- Oracle Database Cloud Service
- Microsoft Azure SQL Database
- Amazon Relational Database Services (RDS)

Available as:

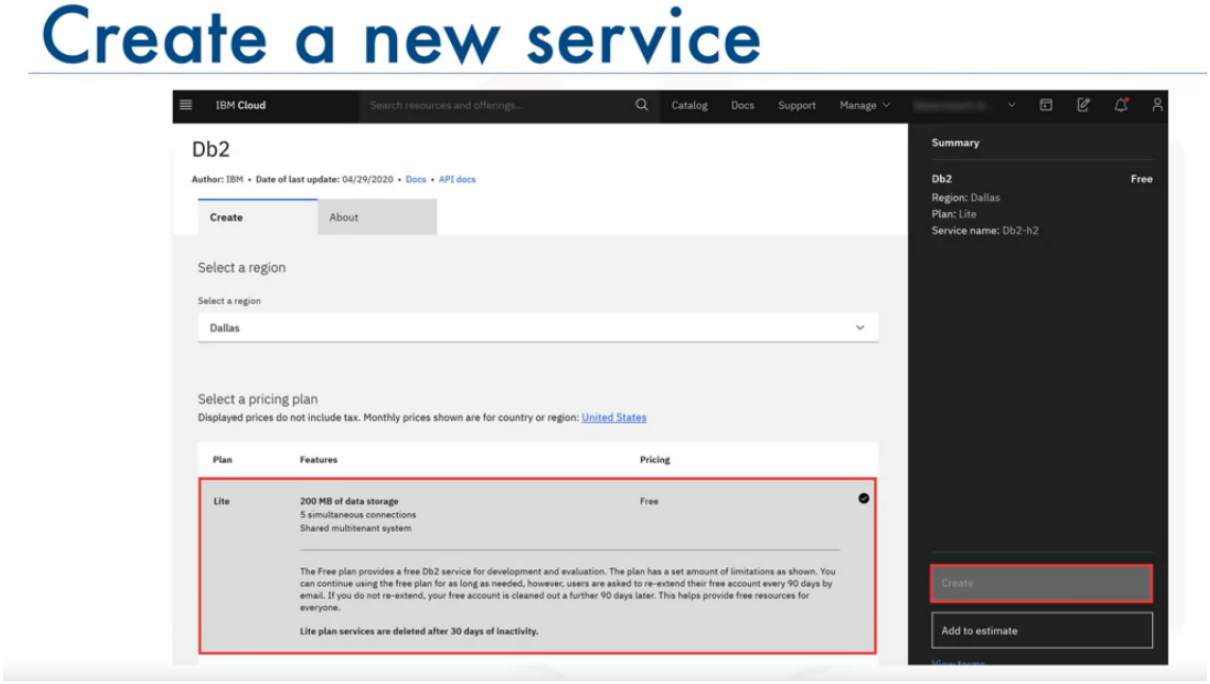
- VMs or Managed Service
- Single or Multi-tenant



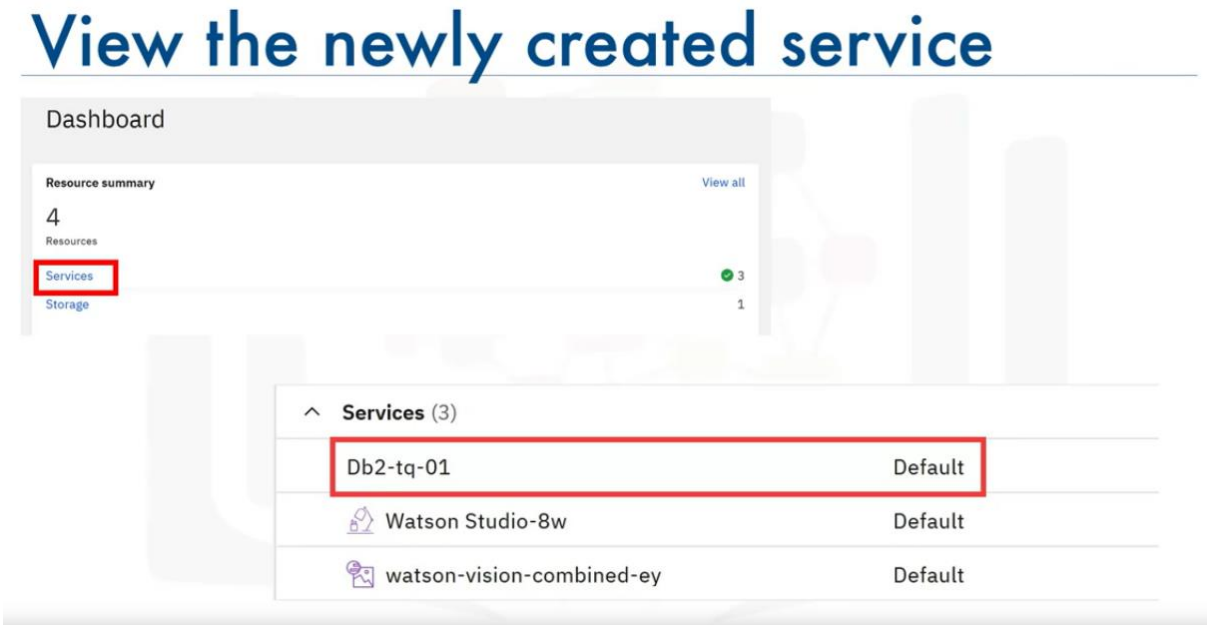


1. You can use Db2 on Cloud just as you would use any database software, but without the overhead and expensive hardware setup or software installation and maintenance. Now let's see how we can set up a service instance of Bb2.

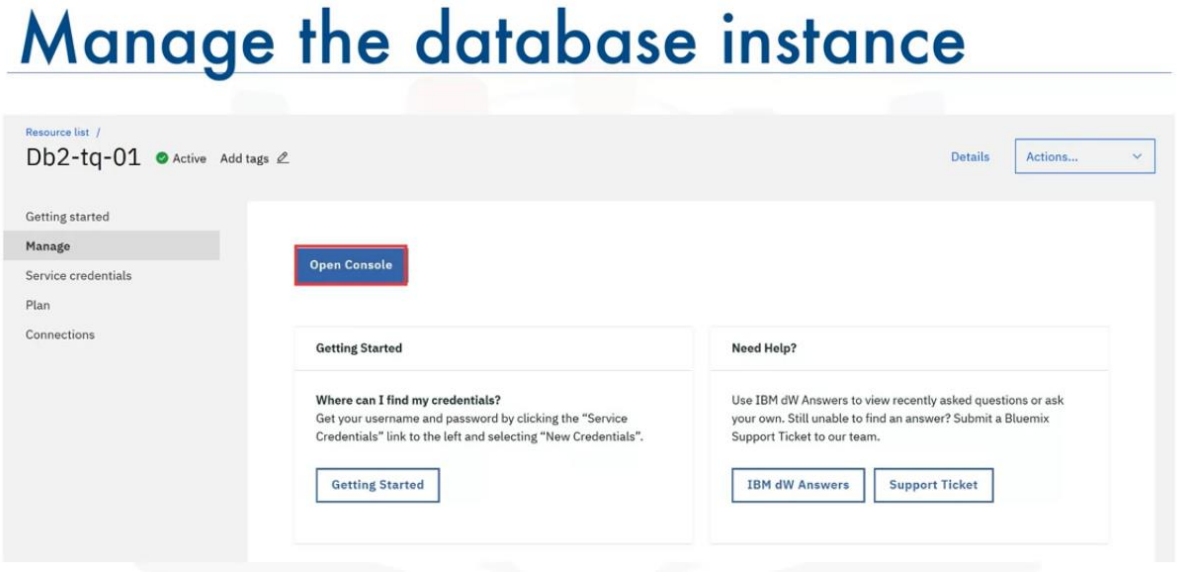
2. Navigate to IBM Cloud catalog and select the Db2 service. Note there are several variations of the Db2 service, including Db2 Hosted and Db2 Warehouse. For our purposes, we will choose the Db2 service which comes with a free lite plan.



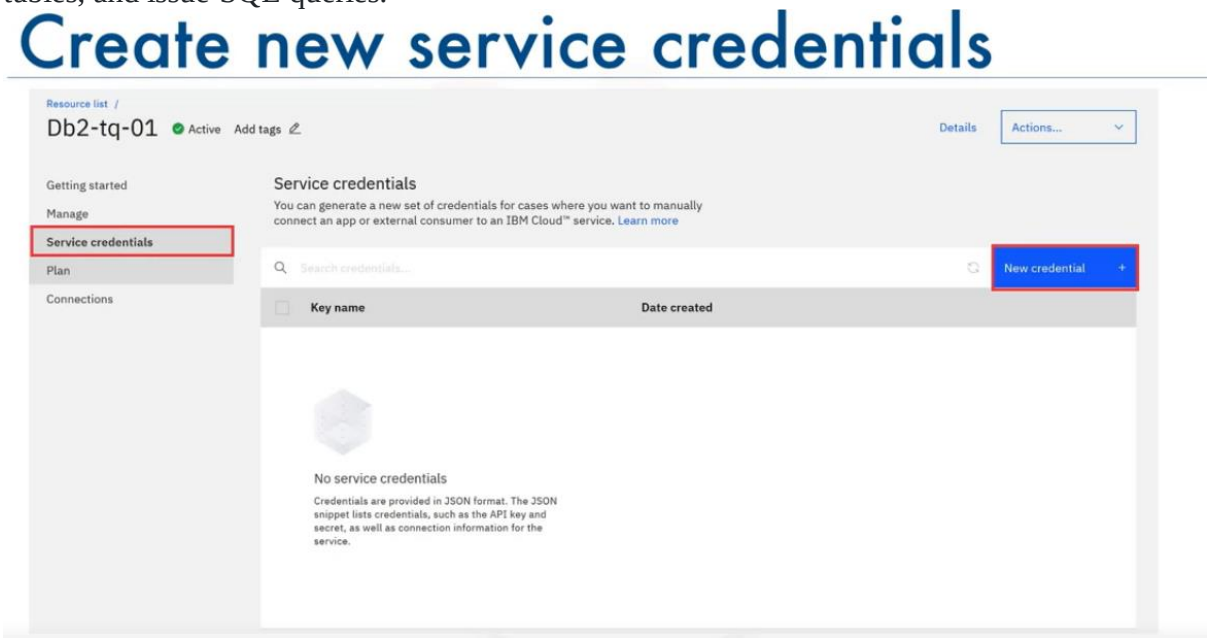
3. Select the lite plan. If need to, change the defaults. You can type a service instance name, choose the region to deploy to, as well as an org and space for the service, then click "Create".



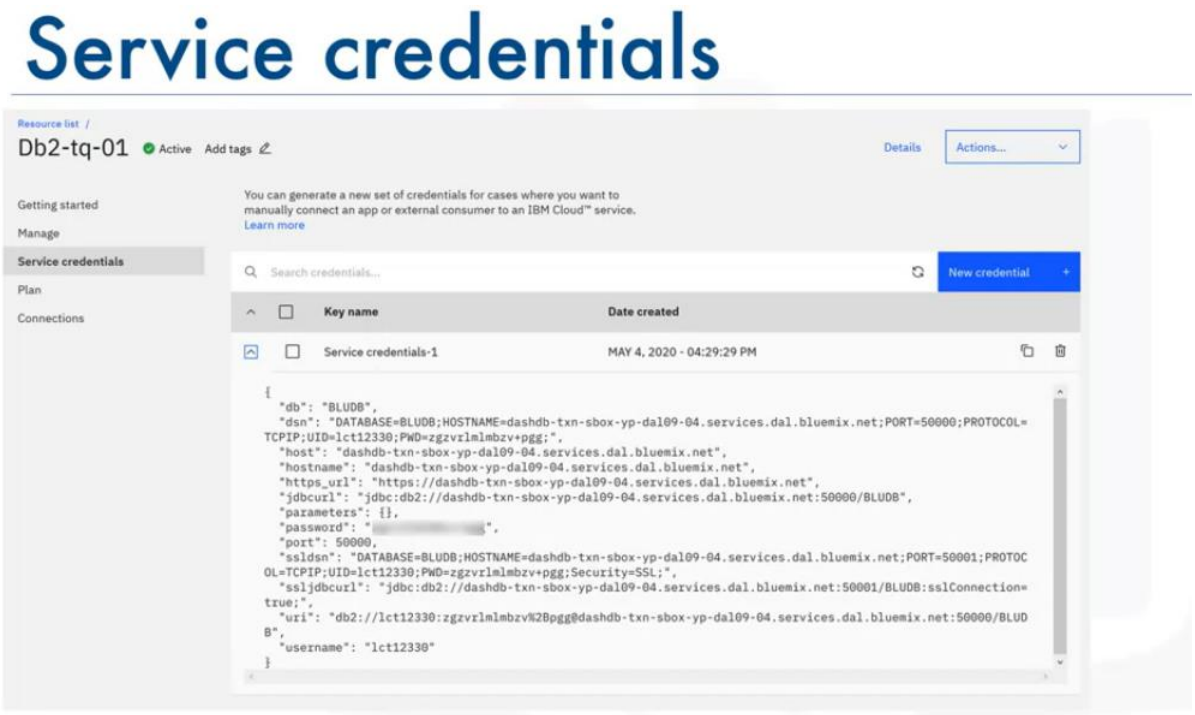
4. You can view the IBM Db2 service that you created by selecting services from your IBM Cloud dashboard.

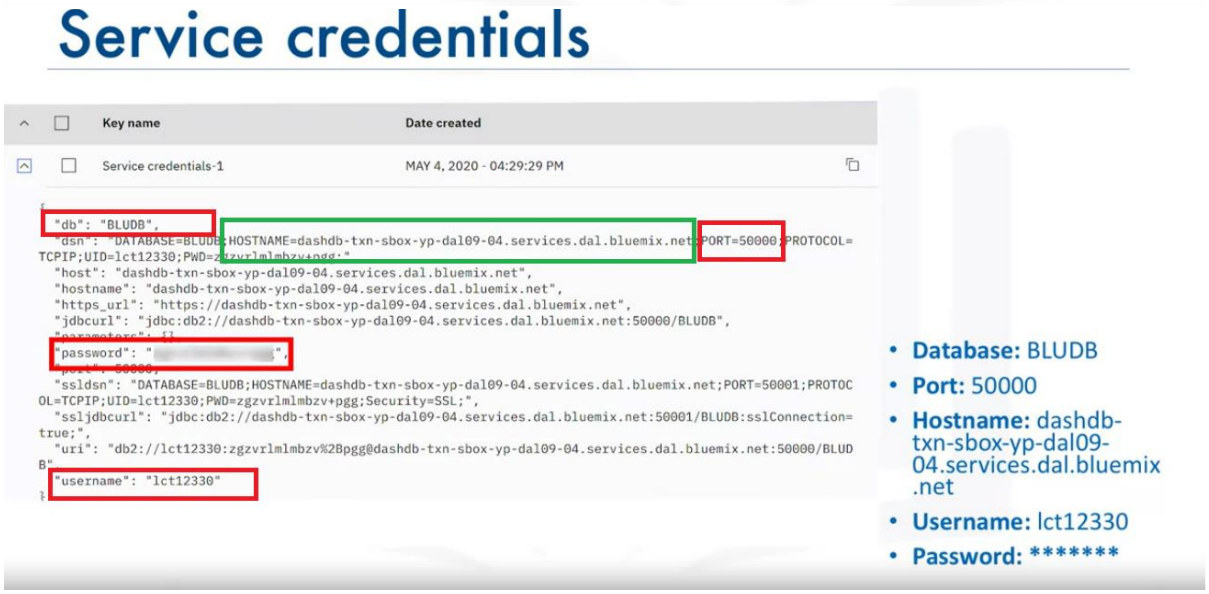


5. From this dashboard, you can manage your database instance. For example, you can click on the "Open Console" button to launch the Web Console for your database instance. The Web Console allows you to create tables, load data, explore data in your tables, and issue SQL queries.



6. In order to access your database instance from your applications, you will need the service credentials. For the first time around, you'll need to create a set of new credentials. You can also choose to create multiple sets of credentials for different applications and users.





7. Once a set of service credentials is created, you can view it as adjacent snippet. The credentials include the necessary details to establish a connection to the database, and includes the following; a database name and port number, a host name, which is the name of the server on the Cloud on which your database instance resides, a username, which is the user ID you'll use to connect along with the password. Note that your username is also the schema name in which your tables will be created by default. Now that you know how to create a database instance on Cloud, the next step is to actually go and create one.

Hands-on Lab: Sign up for IBM Cloud, Create Db2 service instance and get started with the Db2 console

From now on, the hands-on labs for this course require an environment for working with a relational database. To get you up and running quickly we will do so on the Cloud, so you don't have to worry about downloading or installing anything, rather, simply access your database from your web browser. IBM Cloud provides a large number of Data and Analytics services, including IBM Db2, a next generation SQL database. And best of all you can create a Lite account on IBM Cloud and provision an instance of Db2 (Lite Plan) at no charge, and without providing any credit card information.

Objectives

After completing this lab, you will be able to:

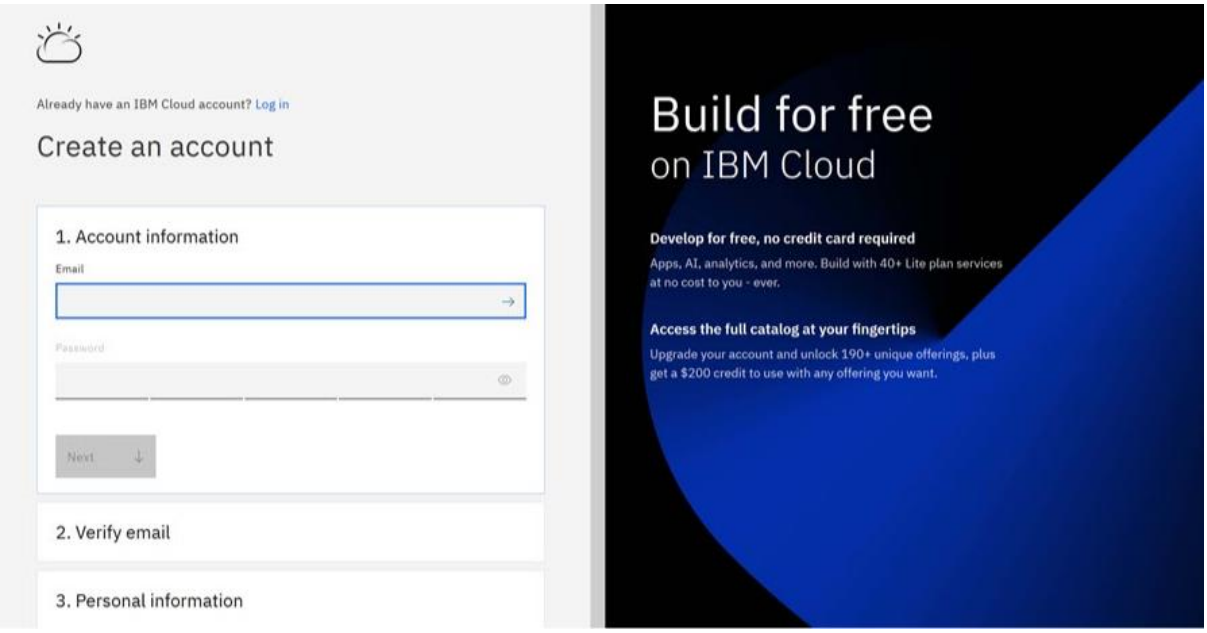
- Sign up for IBM Cloud
- Create an instance of a Db2 service
- Locate and explore the Db2 console

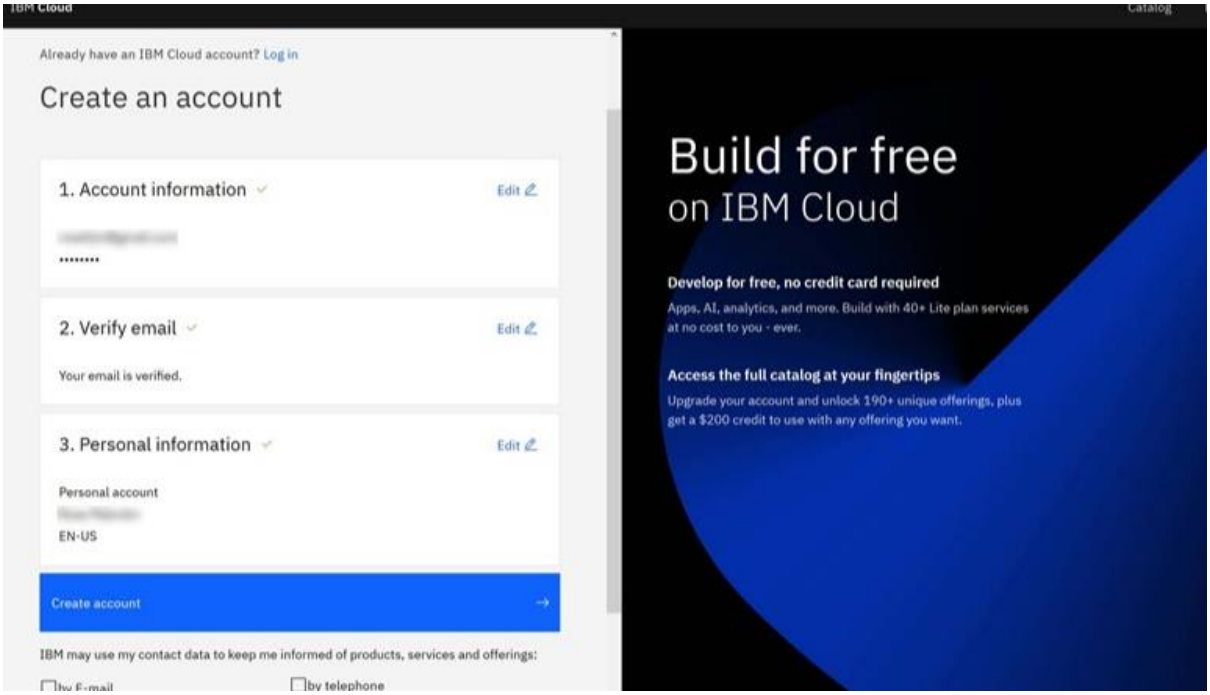
Exercise 1: Sign up for IBM Cloud

First, let us introduce you to the IBM Cloud. IBM Cloud is IBM's innovative cloud computing platform which combines platform as a service (PaaS) with infrastructure as a service (IaaS). It includes a rich catalog of cloud services that can be easily integrated with PaaS and IaaS to build business applications rapidly. It lets you quickly create, deploy, and manage your applications in the cloud. You don't have to deal with any of the underlying infrastructure. And you don't have to deal with the hosting and the managing of your cloud-based applications. You can use the tools and languages of your choice. IBM Cloud comes with a catalog full of awesome services, or APIs, from IBM and from third parties that you can use in your applications. You can scale your applications with literally a couple of clicks.

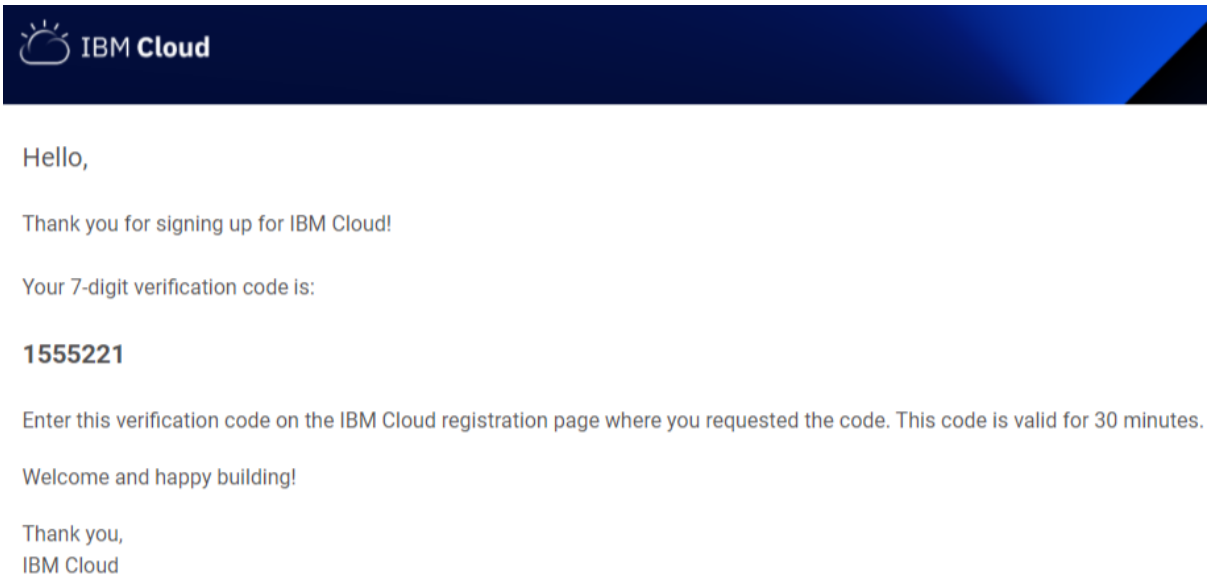
Please follow the steps given below to create an IBM Cloud account.

1. Sign up for IBM Cloud at: cloud.ibm.com/registration
    - **Note**: If you already have an IBM Cloud account (perhaps you created it for another course), you can skip this step and simply log in. If your older IBM Cloud account trial has expired and is asking for a credit card to upgrade, you can instead create a new IBM Cloud account with a different email address.





2. As part of the sign-up process, you will receive an email with a verification code in your inbox. Enter this verification code on the IBM Cloud registration page where you requested the code, to activate your account.



3. If needed, or if you have already signed up for IBM Cloud, login to IBM Cloud at: cloud.ibm.com/login

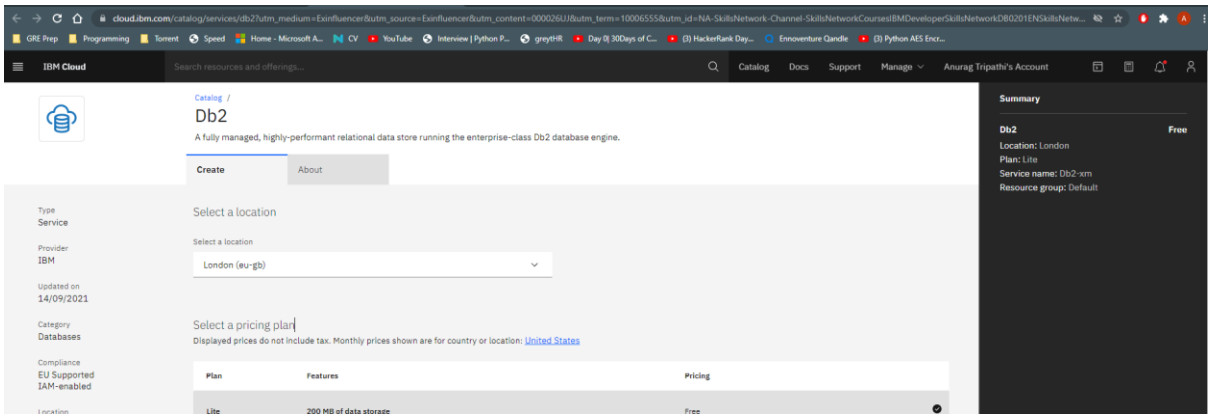
Exercise 2: Create an instance of IBM Db2 Lite plan

Now let us introduce you to Db2 on IBM Cloud. IBM Db2 is a next generation SQL database provisioned for you in the cloud. You can use Db2 on IBM Cloud just as you would use any database software (RDBMS), but without the overhead and expense of hardware setup or software installation and maintenance. Among the service plans offered for Db2 on IBM Cloud is the Lite plan, which is free to use. You can use your database instance to store relational data, analyse data using a built-in SQL editor, or by connecting your own apps.

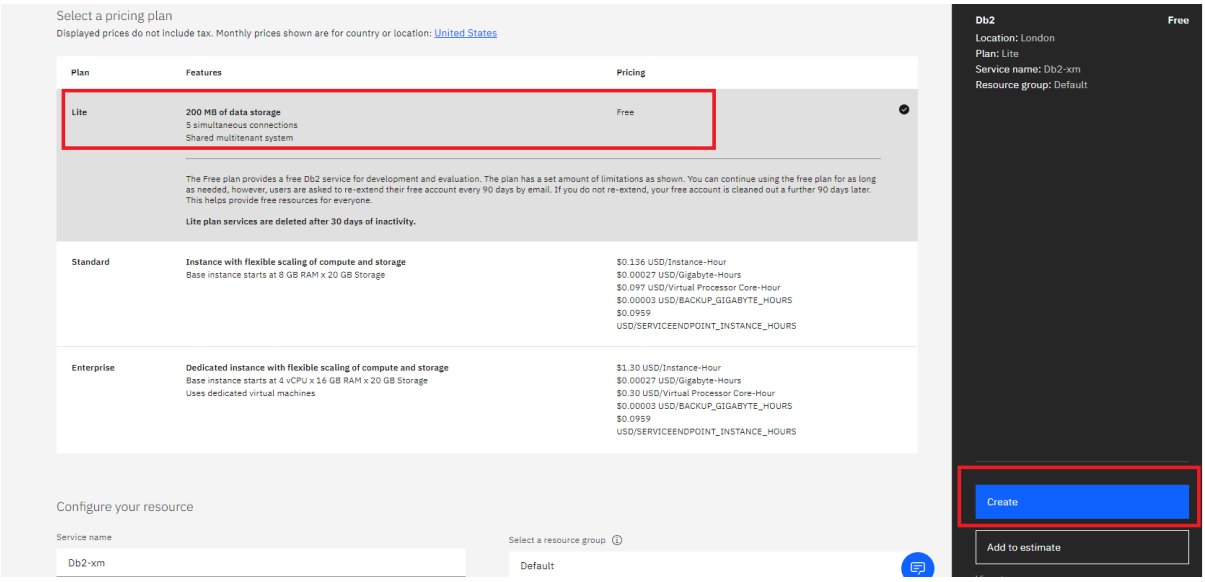
Note that IBM Cloud also provides other variants of Db2 such as Db2 Hosted and Db2 Warehouse on Cloud, which is also referred to in this course. However, for the labs in this course, we will utilize the Db2 service since it comes with a Lite plan which is free to use.

Please follow the steps given below to provision an instance of Db2 on IBM Cloud.

1. Ensure you are logged in to IBM Cloud and go to: cloud.ibm.com/catalog/services/db2
    - **Note**: Depending on the Country of your IBM Cloud account a region/location to deploy will be pre-selected. For example, if you are in the US, the default Datacenter location will be Dallas. Users from the UK will see London and so on. It is best to go with the default location that is closest to you.



2. Scroll down to the Pricing Plans section and select the **Lite plan** (it's a free plan and does not require a credit card).
3. Then click on the **Create** button towards the lower-right of the page. It will spin for a few seconds (typically less than 30s) and then you should see a Service Created message indicating that your instance of Db2 was created successfully.

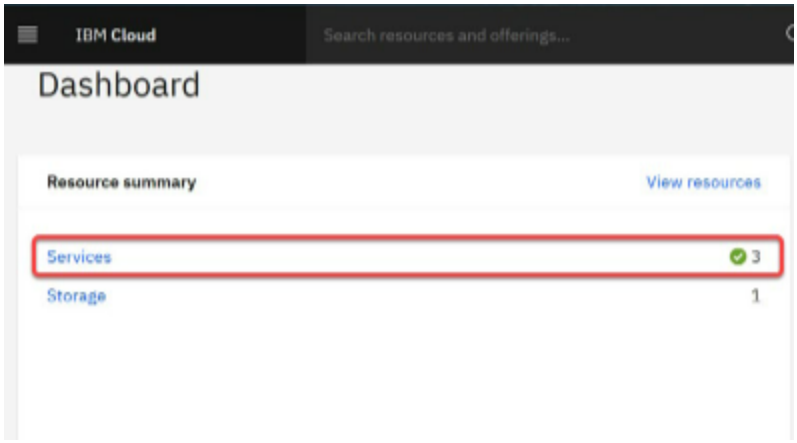


Exercise 3: Locate and Explore the Db2 console

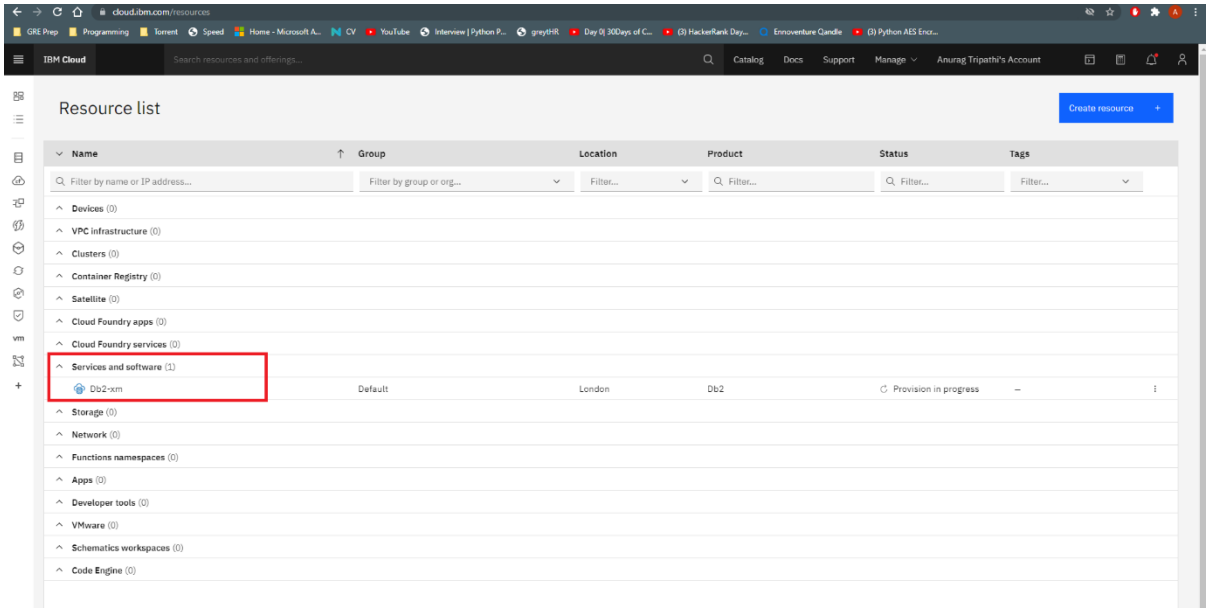
Now that you have created your database instance, you need to know how to get to it, explore the console and start working with it.

- **NOTE:** You are not required to compose and run any SQL query on this exercise.

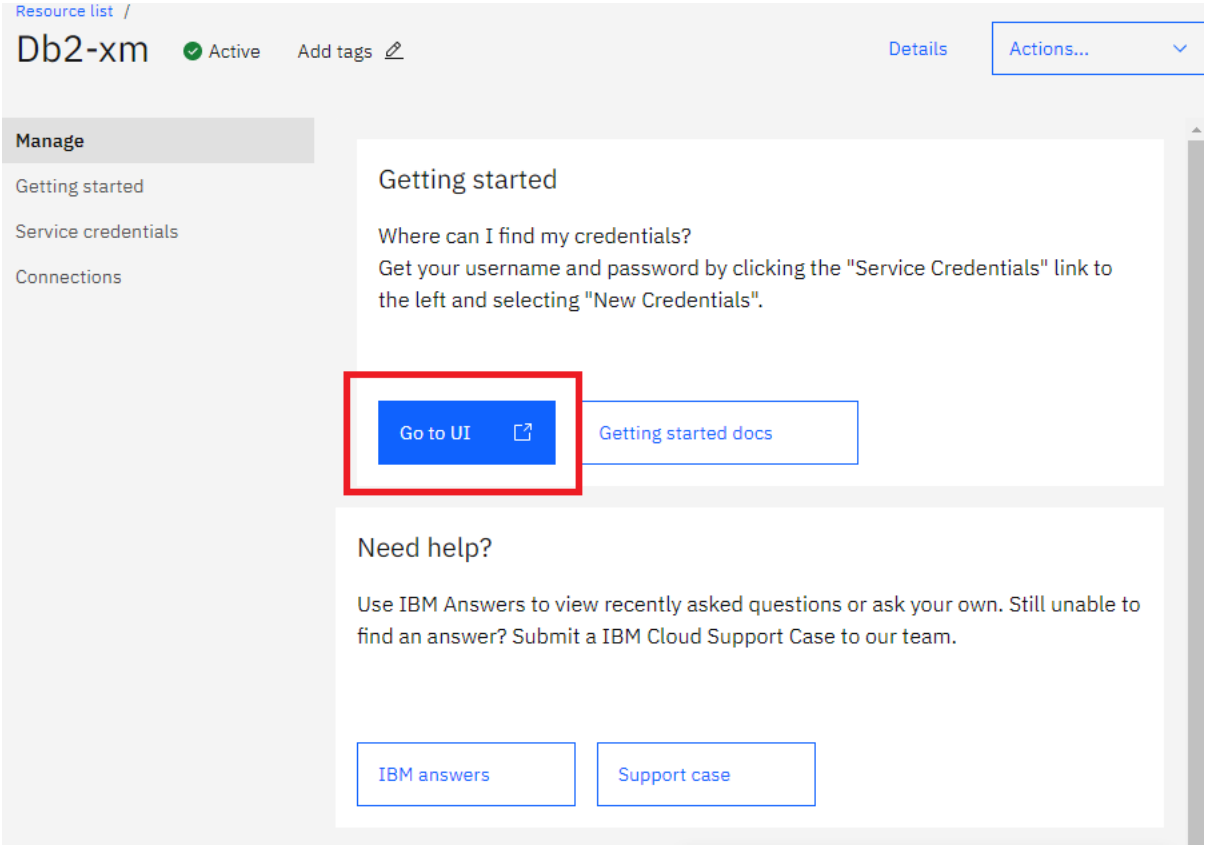
1. To access your database instance, go to your IBM Cloud Resource List (you may need to log into IBM Cloud in the process) directly at: cloud.ibm.com/resources
    - **Alternative**: Go to your IBM Cloud dashboard (you may need to login to IBM Cloud in the process) at: cloud.ibm.com and click **Services**.



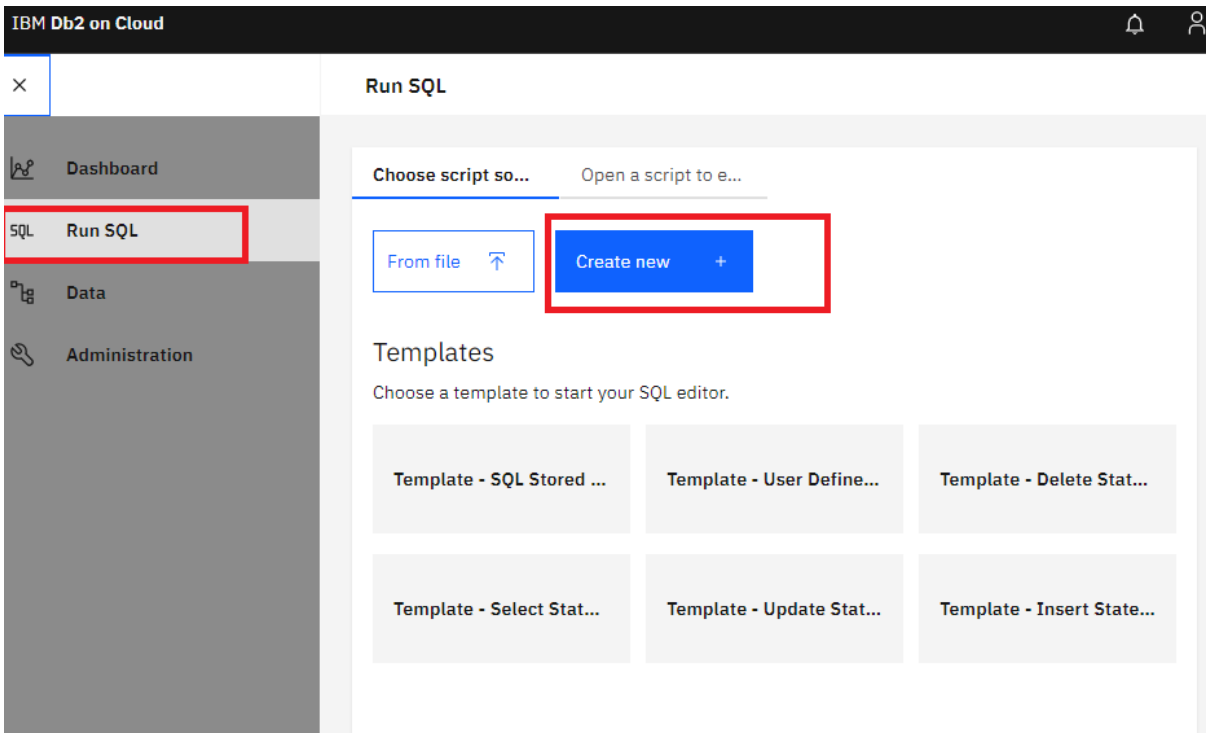
2. In the Resource list, expand the **Services** and locate and click on your instance of Db2 you provisioned in exercise 2 (the name typically starts with Db2-xx for example Db2-fk, Db2-50, etc.)



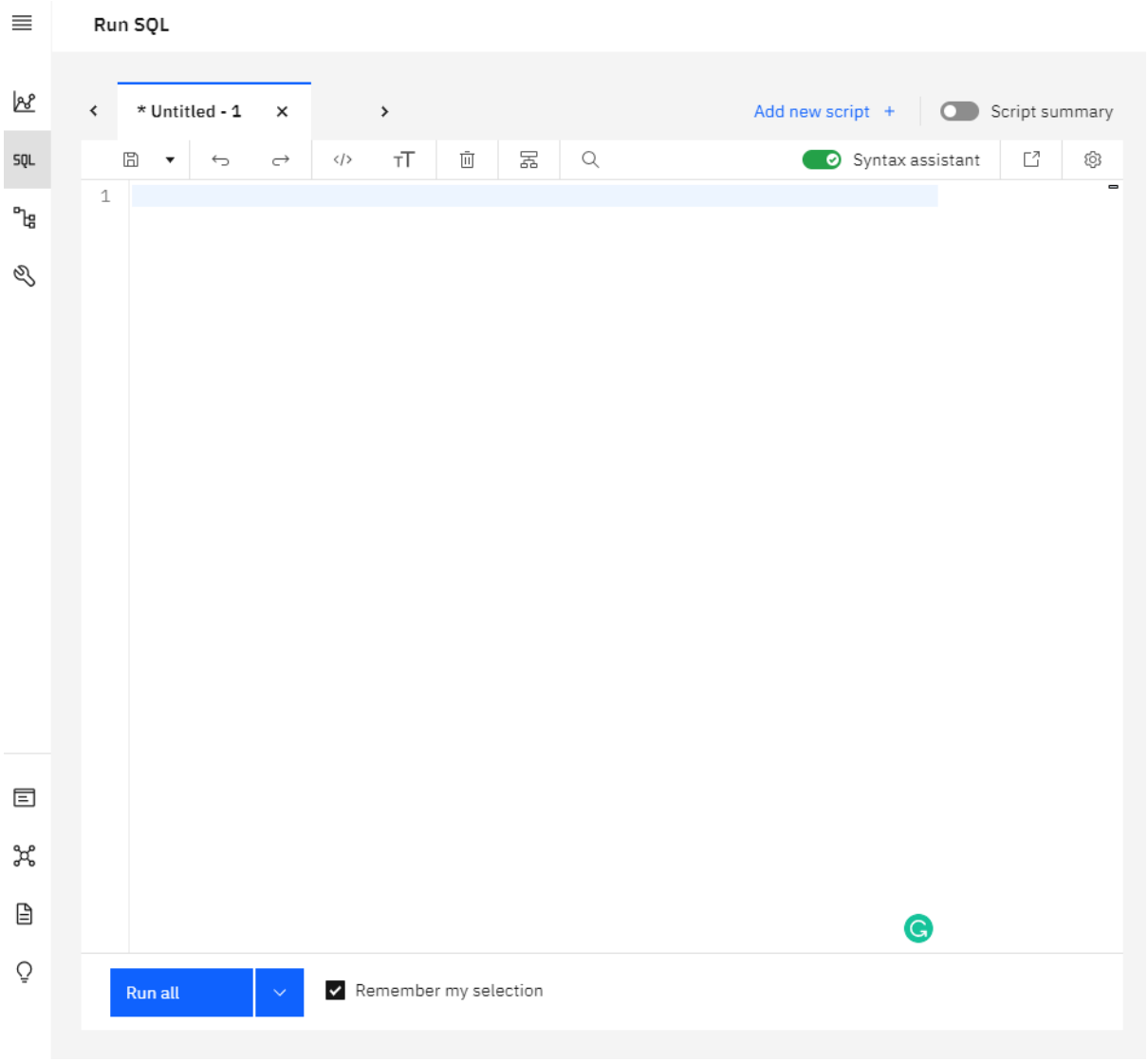
3. Click on the **Go to UI** button.



4. 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**.

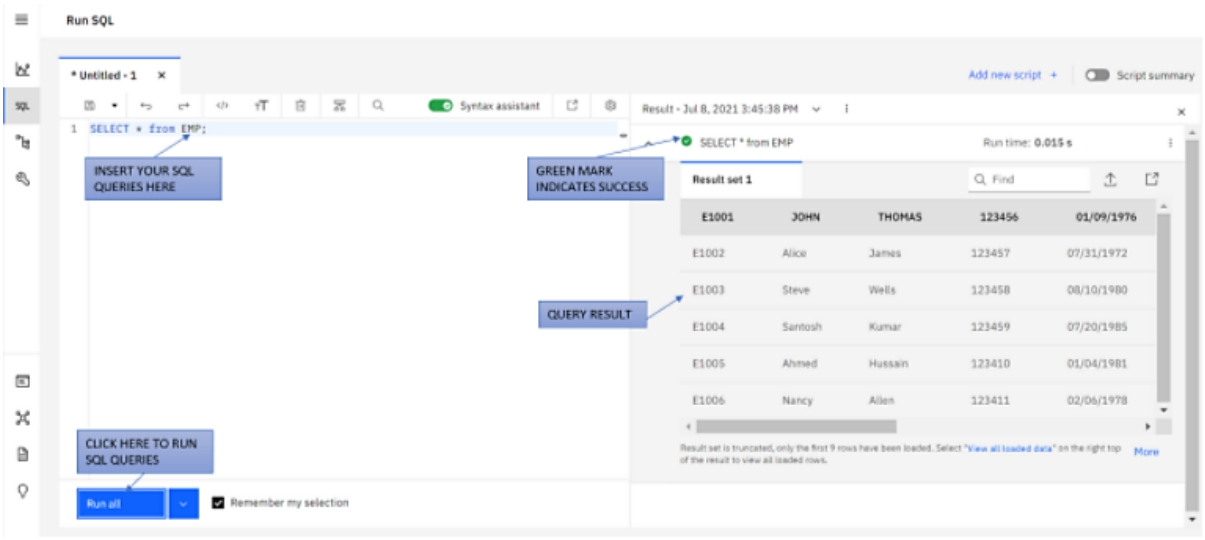


5. On the next screen click on **Blank (Create new)**.

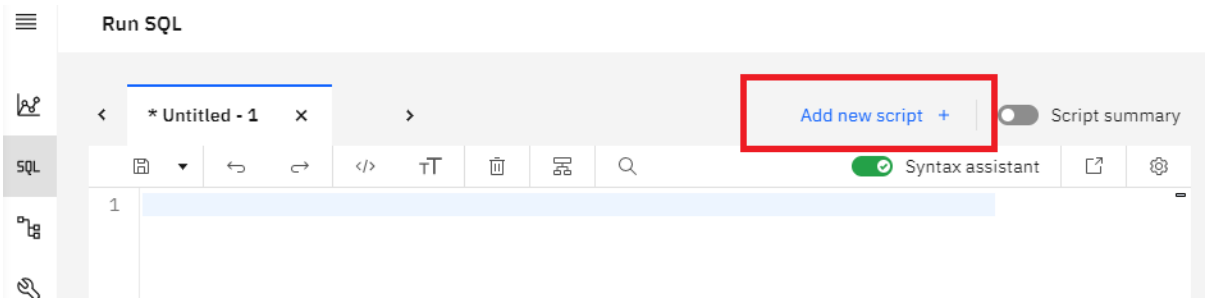


6. The SQL editor will open where you can start typing and running queries.

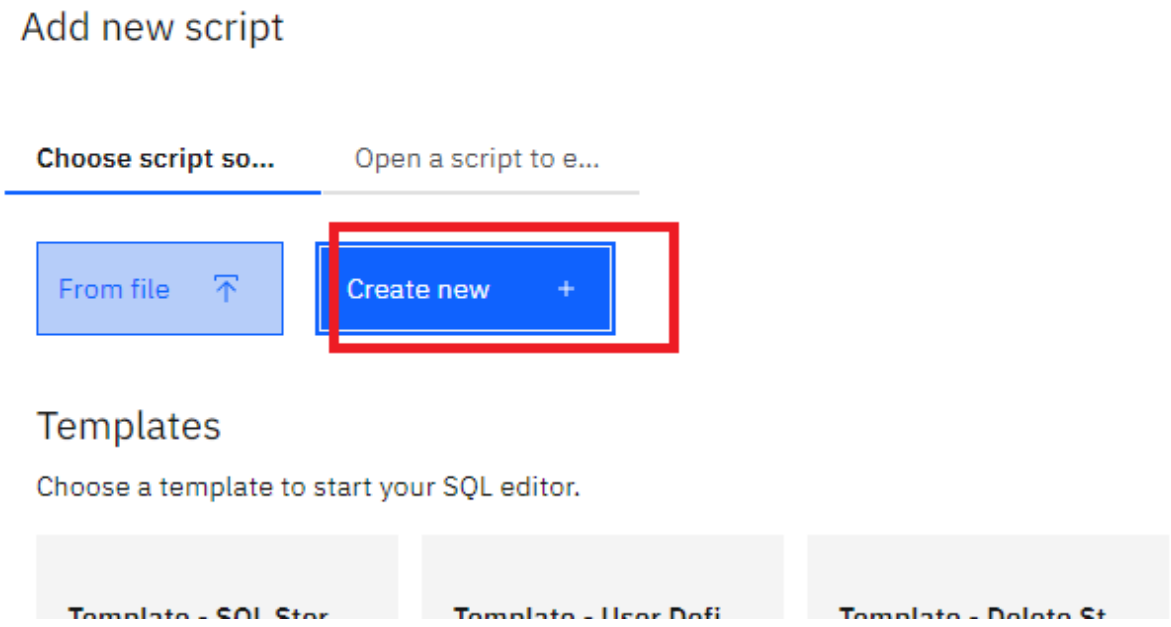
7. The SQL editor has several areas for performing different tasks.



8. Click on the **Add New Script** + icon if you want to add a new script for composing queries.

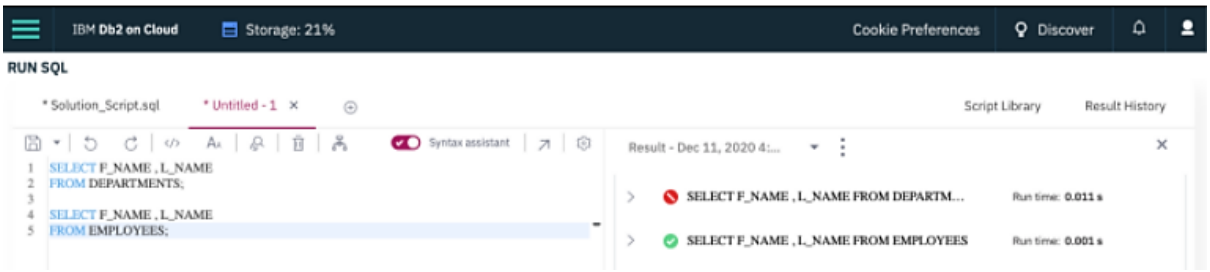


9. Click on **Blank**.

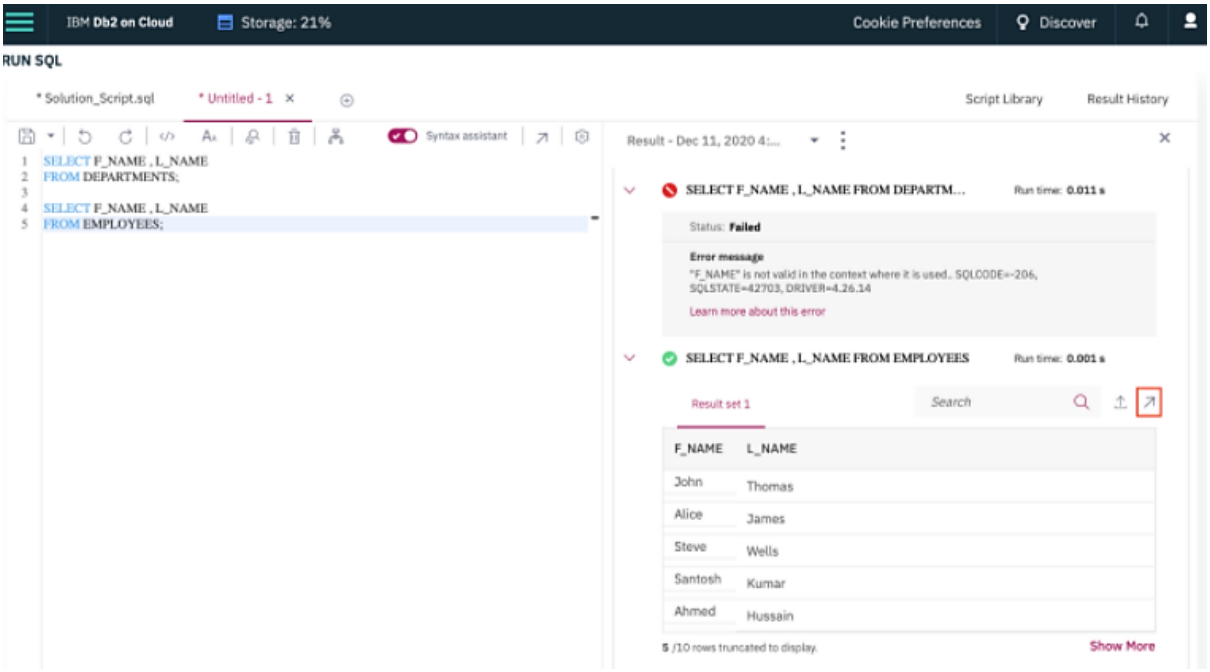


10. When you are asked in the upcoming labs, compose the appropriate SQL query for each problem and run by clicking **Run all** .

11. When you will run the script, looking at the Result section of the executed queries you will know whether the SQL statements ran successfully or not.



12. By clicking the Result section of the executed queries, you can see the query error details or result set to check and ensure the output is what you expected.
    - **NOTE:** You may find that some results don't appear in the result set pane; in this case, click the highlighted **diagonal arrow (View More)** and it will open the full Result Set window containing the results.



Summary

You can now find your way into and around the database instance, and you will use these skills in later labs.

Types of SQL statements (DDL vs. DML)

Types of SQL Statements - DDL

- SQL Statement types: DDL and DML
- DDL (Data Definition Language) statements:
  - Define, change, or drop data
- Common DDL:
  - CREATE
  - ALTER
  - TRUNCATE
  - DROP
- DML (Data Manipulation Language) statements:
  - Read and modify data
  - CRUD operations (Create, Read, Update & Delete rows)
- Common DML:
  - INSERT
  - SELECT
  - UPDATE
  - DELETE

Summary

Now you know that:

- DDL used for defining objects (tables)
- DML used for manipulating data in tables

CREATE TABLE Statement

Objectives

At the end of this video, you will be able to:

- Create a Table in a relational database using Entity Name, Attributes and the CREATE TABLE statement

- **Syntax:**

```
CREATE TABLE table_name
    (
     column_name_1 datatype optional_parameters,
     column_name_2 datatype,
     ...
     column_name_n datatype
    )
```

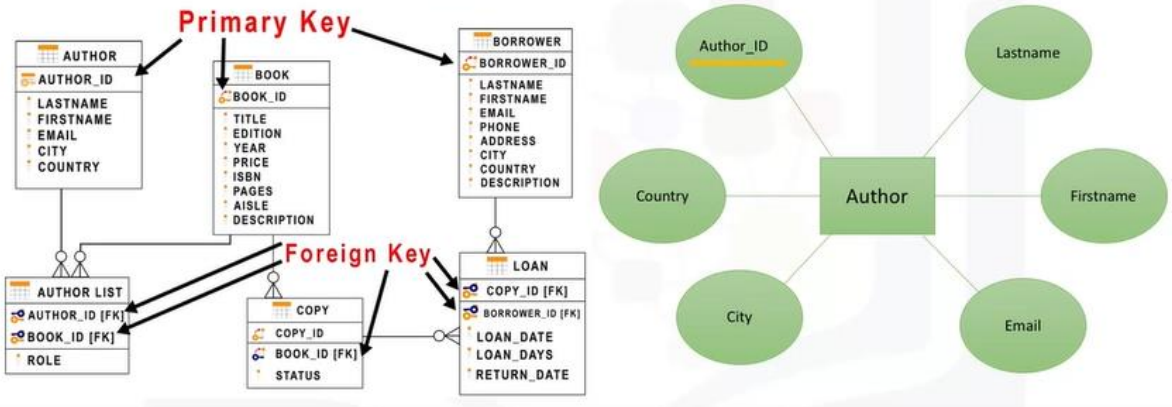
EXAMPLE

- Create a table for Canadian provinces

```
CREATE TABLE provinces(
    id char(2) PRIMARY KEY NOT NULL,
    name varchar(24)
   )
```

id *char(2)*	name *varchar(24)*
AB	ALBERTA
BC	BRITISH COLUMBIA
...	...

Create a table



Primary Key: Uniquely Identifies each Row in a Table

- Now, let's look at a more elaborate example based on the Library database. This database includes several entities such as AUTHOR, BOOK, BORROWER, etc.
- Let's start by creating the table for the AUTHOR entity.
- The name of the table will be AUTHOR, and its attributes such as AUTHOR_ID, FIRSTNAME, LASTNAME, etc. will be the columns of the table.
- In this table, we will also assign the Author_ID attribute as the Primary Key, so that no duplicate values can exist. Recall, the Primary Key of a relational table uniquely identifies each tuple (or row) in a table.
- To create the Author table, issue the following command:

```
CREATE TABLE author (
    author_id CHAR(2) PRIMARY KEY NOT NULL,
    lastname VARCHAR(15) NOT NULL,
    firstname VARCHAR(15) NOT NULL,
    email VARCHAR(40),
    city VARCHAR(15),
    country CHAR(2)
  )
```

CREATE TABLE author (author_id CHAR(2) PRIMARY KEY NOT NULL, lastname VARCHAR(15) NOT NULL, firstname VARCHAR(15) NOT NULL, email VARCHAR(40), city VARCHAR(15), country CHAR(2))

- Note that the Author_ID is the Primary Key. This constraint prevents duplicate values in the table.
- Also note that Last Name and First Name have the constraint NOT NULL. This ensures that these fields cannot contain a NULL value, since an author must have a name.

ALTER, DROP, and Truncate tables

Objectives

After watching this video, you will be able to:

- Describe the ALTER TABLE, DROP TABLE, and TRUNCATE statements
- Explain the syntax
- Use the statement in queries

ALTER TABLE ... ADD COLUMN

- Add or remove columns
- Modify the data type of columns
- Add or remove keys
- Add or remove constraints

```
ALTER TABLE <table_name>
  ADD COLUMN <column_name_1> datatype
  . . .
  ADD COLUMN <column_name_n> datatype;
```

```
ALTER TABLE author
  ADD COLUMN telephone_number BIGINT;
```

author_id	lastname	firstname	email	city	country	telephone_number
1001	Thomas	John	johnt@...	New York	USA	5551111
1002	James	Alice	alicej@...	Seattle	USA	5551112
1003	Wells	Steve	stevew:@...	Montreal	Canada	5552222
1004	Kumar	Santosh	kumars@...	London	UK	5553333

In this example, the data type for the column is BIGINT which can hold a number up to 19 digits long.

```
ALTER TABLE <table_name>
  ALTER COLUMN <column_name> SET DATA TYPE
<datatype>;
```

You also use the ALTER TABLE statement to modify the data type of a column. To do this, use the ALTER COLUMN clause specifying the new data type for the column.

```
ALTER TABLE author
  ALTER COLUMN telephone_number SET DATA TYPE
CHAR(20);
```

author_id	lastname	firstname	email	city	country	telephone_number
1001	Thomas	John	johnt@...	New York	USA	555-1111
1002	James	Alice	alicej@...	Seattle	USA	555-1112
1003	Wells	Steve	stevew@...	Montreal	Canada	555-2222
1004	Kumar	Santosh	kumars@...	London	UK	555-3333

Using a numeric data type for telephone number means that you cannot include parentheses, plus signs, or dashes as part of the number. You can change the column to use the CHAR data type to overcome this.

author_id	lastname	firstname	email	city	country	telephone_number
1001	Thomas	John	johnt@...	New York	USA	555-1111
1002	James	Alice	alicej@...	Seattle	USA	555-1112
1003	Wells	Steve	stevew@...	Montreal	Canada	555-2222
1004	Kumar	Santosh	kumars@...	London	UK	555-3333

Altering the data type of a column containing existing data can cause problems though if the existing data is not compatible with the new data type. For example, changing a column from the CHAR data type to a numeric data type will not work if the column already contains non-numeric data. If you try to do this, you will see an error message in the notification log and the statement will not run. If your spec changes and you no longer need this extra column, you can again use the ALTER TABLE statement, this time with the DROP COLUMN clause, to remove the column.

```
ALTER TABLE author
    DROP COLUMN telephone_number;
```

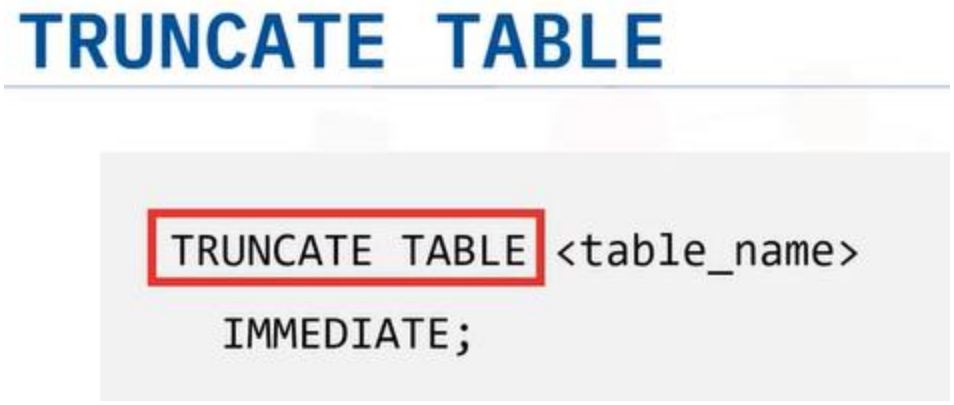
author_id	lastname	firstname	email	city	country	telephone_number
1001	Thomas	John	johnt@...	New York	USA	555-1111
1002	James	Alice	alicej@...	Seattle	USA	555-1112
1003	Wells	Steve	stevew:@...	Montreal	Canada	555-2222
1004	Kumar	Santosh	kumars@...	London	UK	555-3333

DROP TABLE

```
DROP TABLE <table_name>;
```

```
DROP TABLE author;
```

Sometimes you might want to just delete the data in a table rather than deleting the table itself. While you can use the DELETE statement without a WHERE clause to do this, it is generally quicker and more efficient to truncate the table instead. You use the TRUNCATE TABLE statement to delete all of the rows in a table. The syntax of the statement is: TRUNCATE TABLE table_name IMMEDIATE.



The IMMEDIATE specifies to process the statement immediately and that it cannot be undone. So, to truncate the author table, you use this statement: TRUNCATE TABLE author IMMEDIATE.

```
TRUNCATE TABLE author
  IMMEDIATE;
```

author_id	lastname	firstname	email	city	country
1001	Thomas	John	johnt@...	New York	USA
1002	James	Alice	alicej@...	Seattle	USA
1003	Wells	Steve	stevew:@...	Montreal	Canada
1004	Kumar	Santosh	kumars@...	London	UK

author_id	lastname	firstname	email	city	country

Hands-on Lab: CREATE, ALTER, TRUNCATE, DROP

How does the syntax of a CREATE statement look?

```
CREATE TABLE table_name(
   column1 data type,
   column2 data type,
   column3 data type,
   ....
);
```

How does the syntax of an ALTER statement look?

```
ALTER TABLE table_name
ADD COLUMN column_name data_type column_constraint;

ALTER TABLE table_name
DROP COLUMN column_name;

ALTER TABLE table_name
ALTER COLUMN column_name SET DATA TYPE data_type;

ALTER TABLE table_name
RENAME COLUMN current_column_name TO new_column_name;
```

How does the syntax of a TRUNCATE statement look?

```
TRUNCATE TABLE table_name;
```

How does the syntax of a DROP statement look?

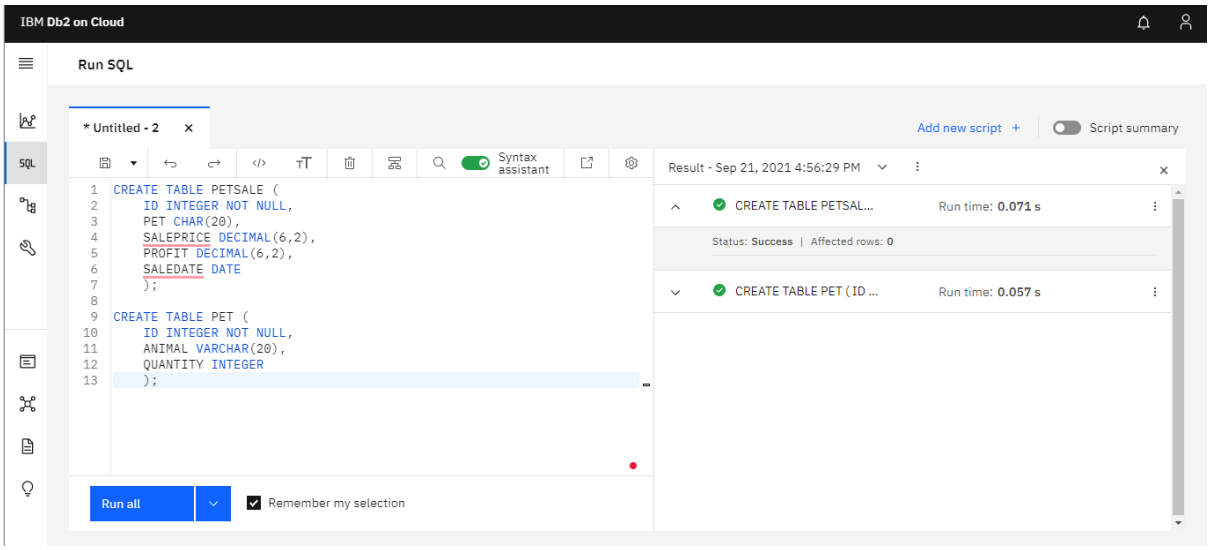
```
DROP TABLE table_name;
```

Exercise 1: CREATE

In this exercise, you will use the CREATE statement to create two new tables using Db2.

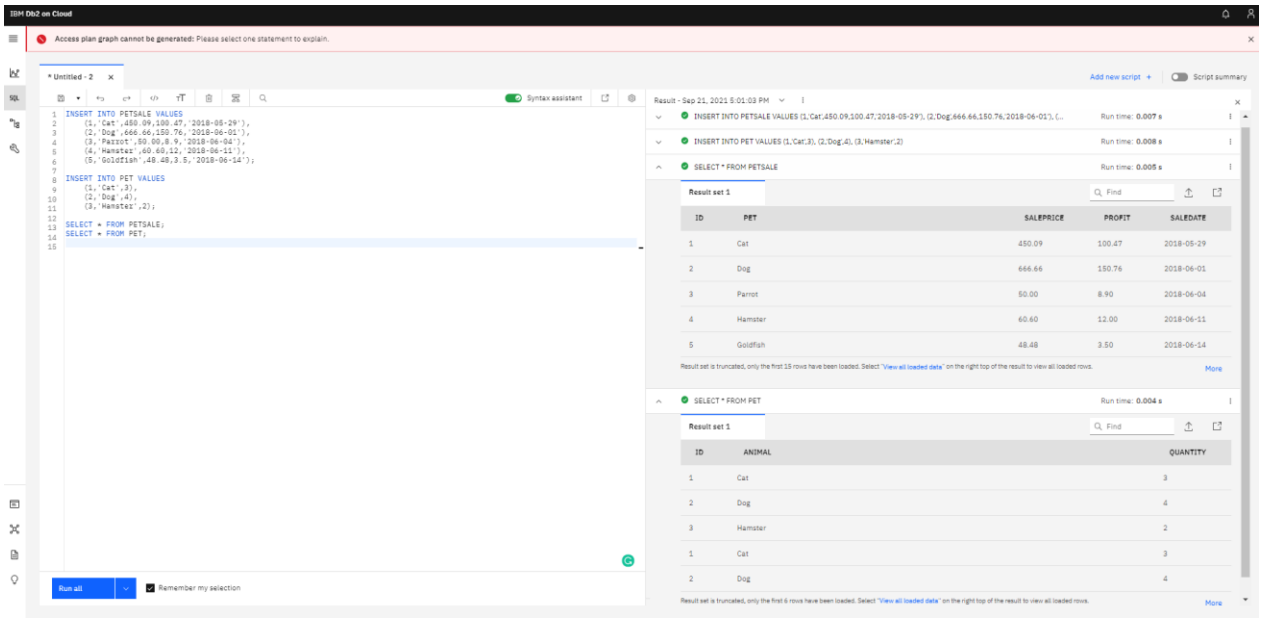
1. You need to create two tables, **PETSALE** and **PET**. To create the two tables PETSALE and PET, copy the code below and paste it to the textbox of the **Run SQL** page. Click **Run all**.

```
CREATE TABLE PETSALE (
                ID INTEGER NOT NULL,
  PET CHAR(20),
  SALEPRICE DECIMAL(6,2),
  PROFIT DECIMAL(6,2),
  SALEDATE DATE
  );

CREATE TABLE PET (
  ID INTEGER NOT NULL,
  ANIMAL VARCHAR(20),
  QUANTITY INTEGER
  );
```



2. Now insert some records into the two newly created tables and show all the records of the two tables. Copy the code below and paste it to the textbox of the **Run SQL** page. Click **Run all**.

```
INSERT INTO PETSALE VALUES
   (1,'Cat',450.09,100.47,'2018-05-29'),
   (2,'Dog',666.66,150.76,'2018-06-01'),
   (3,'Parrot',50.00,8.9,'2018-06-04'),
   (4,'Hamster',60.60,12,'2018-06-11'),
   (5,'Goldfish',48.48,3.5,'2018-06-14');

INSERT INTO PET VALUES
   (1,'Cat',3),
   (2,'Dog',4),
   (3,'Hamster',2);

SELECT * FROM PETSALE;
SELECT * FROM PET;
```



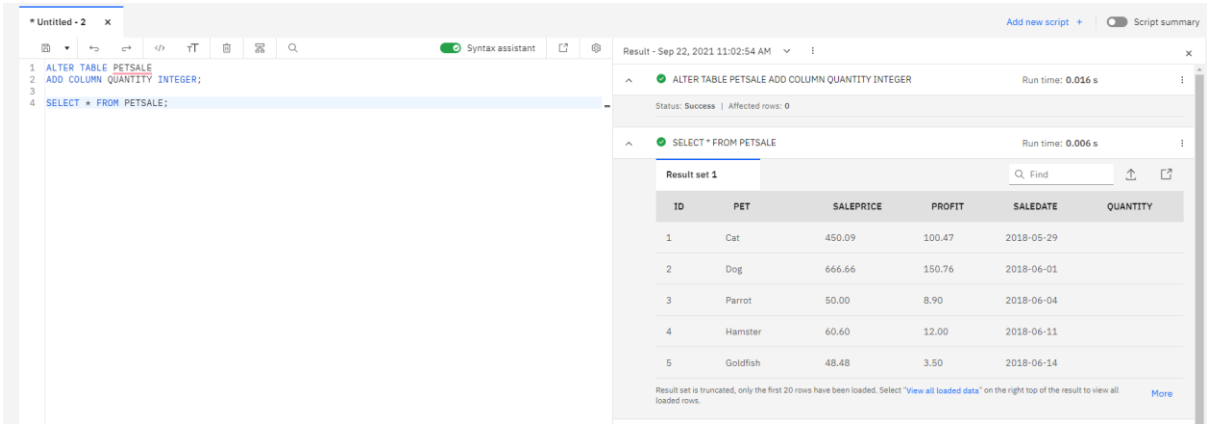
Exercise 2: ALTER

In this exercise, you will use the ALTER statement to add, delete, or modify columns in two of the existing tables created in exercise 1.

Task A: ALTER using ADD COLUMN

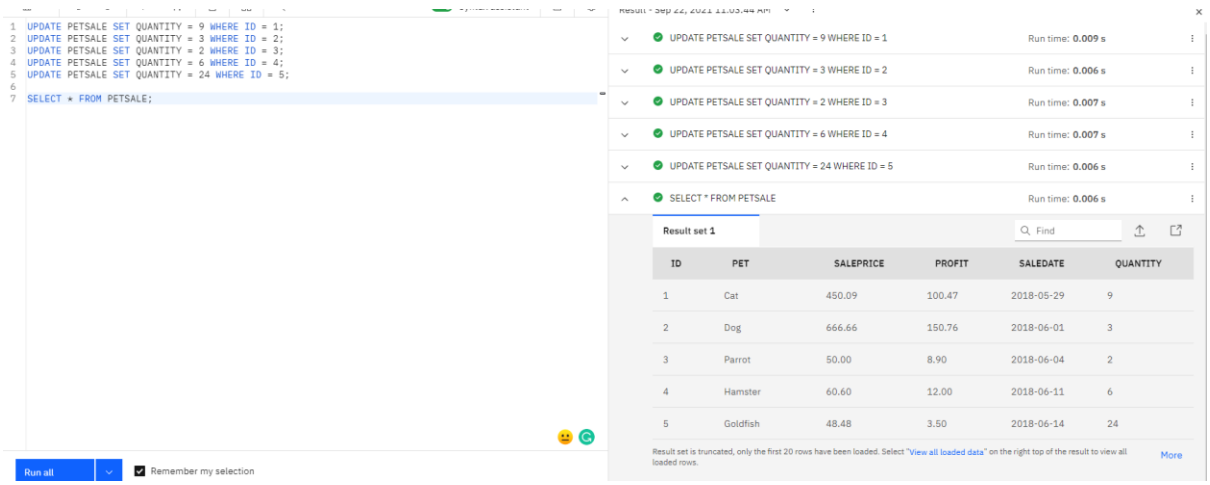
1. Add a new **QUANTITY** column to the **PETSALE** table and show the altered table. Copy the code below and paste it to the textbox of the **Run SQL** page. Click **Run all**.

```
ALTER TABLE PETSALE
ADD COLUMN QUANTITY INTEGER;

SELECT * FROM PETSALE;
```



2. Now update the newly added **QUANTITY** column of the **PETSALE** table with some values and show all the records of the table. Copy the code below and paste it to the textbox of the **Run SQL** page. Click **Run all**.

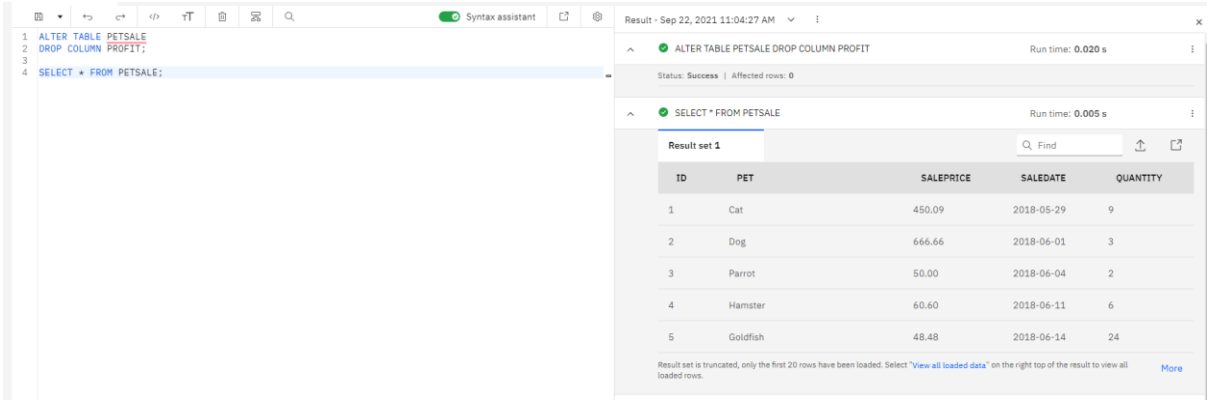
```
UPDATE PETSALE SET QUANTITY = 9 WHERE ID = 1;
UPDATE PETSALE SET QUANTITY = 3 WHERE ID = 2;
UPDATE PETSALE SET QUANTITY = 2 WHERE ID = 3;
UPDATE PETSALE SET QUANTITY = 6 WHERE ID = 4;
UPDATE PETSALE SET QUANTITY = 24 WHERE ID = 5;

SELECT * FROM PETSALE;
```



Task B: ALTER using DROP COLUMN

1. Delete the **PROFIT** column from the **PETSALE** table and show the altered table. Copy the code below and paste it to the textbox of the **Run SQL** page. Click **Run all**.

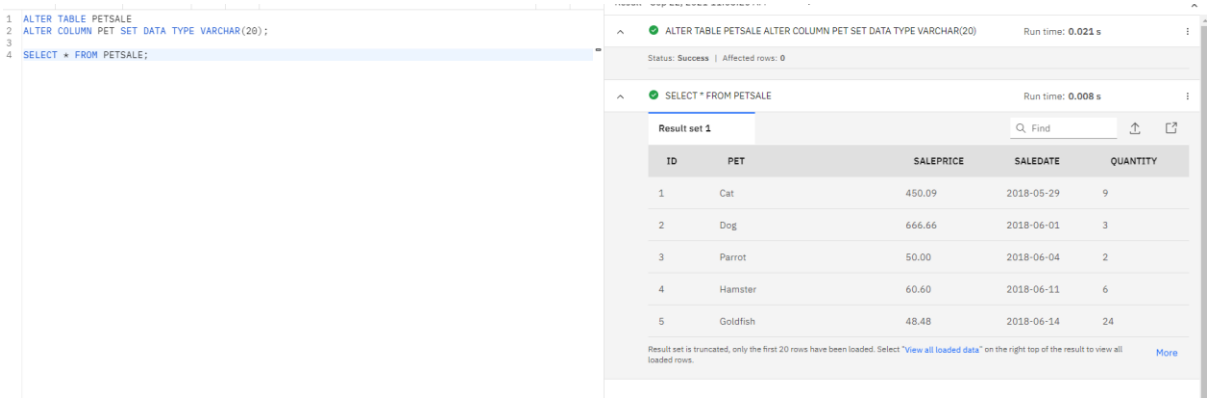
```
ALTER TABLE PETSALE
DROP COLUMN PROFIT;

SELECT * FROM PETSALE;
```



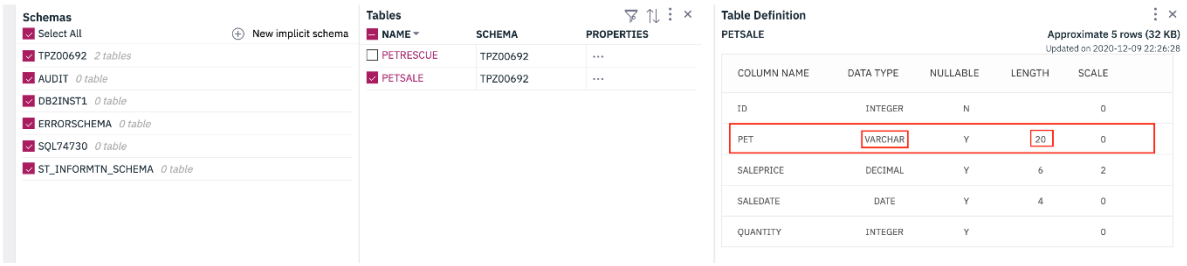
Task C: ALTER using ALTER COLUMN

1. Change the data type to **VARCHAR(20)** type of the column **PET** of the table **PETSALE** and show the altered table. Copy the code below and paste it to the textbox of the **Run SQL** page. Click **Run all**.

```
ALTER TABLE PETSALE
ALTER COLUMN PET SET DATA TYPE VARCHAR(20);

SELECT * FROM PETSALE;
```



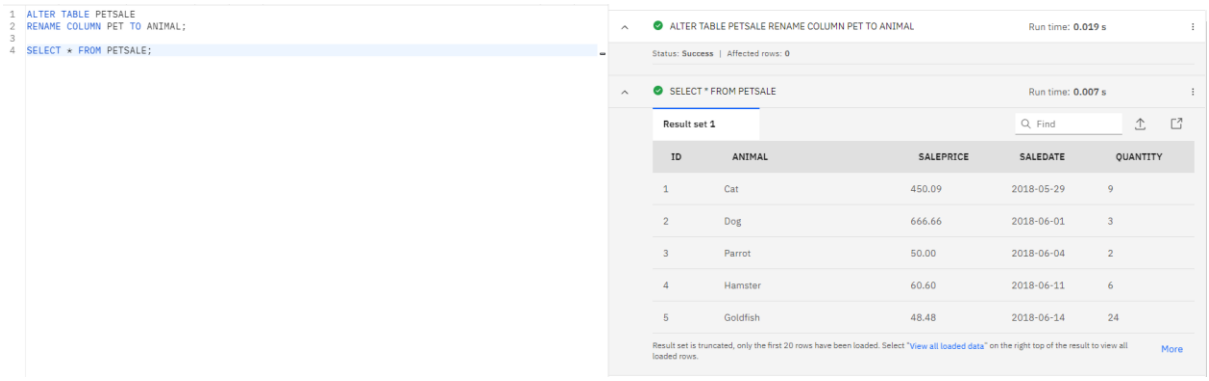
2. Now verify if the data type of the column **PET** of the table **PETSALE** changed to **VARCHAR(20)** type or not. Click on the 3 bar menu icon in the top left corner and click **Explore** > **Tables**. Find the **PETSALE** table from Schemas by clicking **Select All**. Click on the **PETSALE** table to open the Table Definition page of the table. Here, you can see all the current data type of the columns of the **PETSALE** table.



Task D: ALTER using RENAME COLUMN

1. Rename the column **PET** to **ANIMAL** of the **PETSALE** table and show the altered table. Copy the code below and paste it to the textbox of the **Run SQL** page. Click **Run all**.

```
ALTER TABLE PETSALE
RENAME COLUMN PET TO ANIMAL;

SELECT * FROM PETSALE;
```

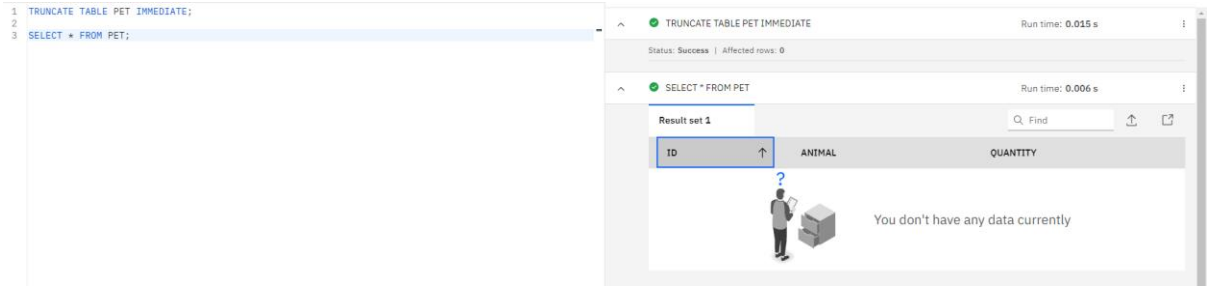


Exercise 3: TRUNCATE

In this exercise, you will use the TRUNCATE statement to remove all rows from an existing table created in exercise 1 without deleting the table itself.

1. Remove all rows from the **PET** table and show the empty table. Copy the code below and paste it to the textbox of the **Run SQL** page. Click **Run all**.

```
TRUNCATE TABLE PET IMMEDIATE;

SELECT * FROM PET;
```

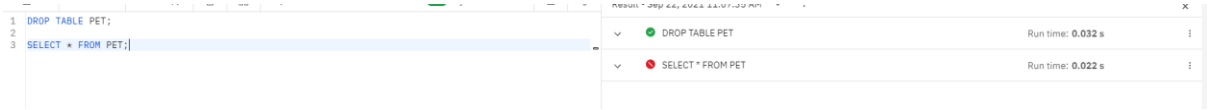


Exercise 4: DROP

In this exercise, you will use the DROP statement to delete an existing table created in exercise 1.

1. Delete the **PET** table and verify if the table still exists or not (SELECT statement won't work if a table doesn't exist). Copy the code below and paste it to the textbox of the **Run SQL** page. Click **Run all**.

```
DROP TABLE PET;

SELECT * FROM PET;
```



Hands-on Lab: Create Tables using SQL Scripts and Load Data into Tables

Objectives

After completing this lab, you will be able to:

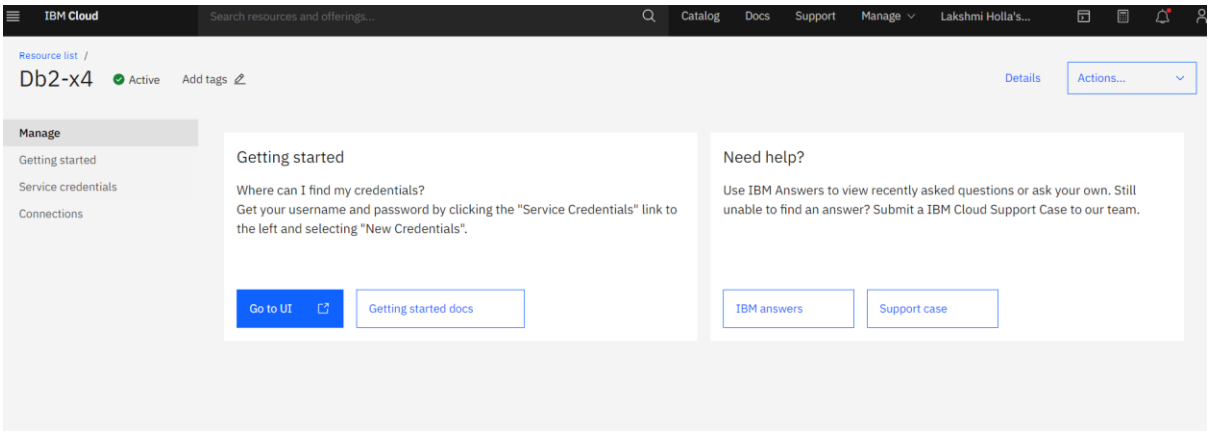
- Create tables using SQL scripts
- Load data into tables

Exercise 1: Create tables using SQL scripts

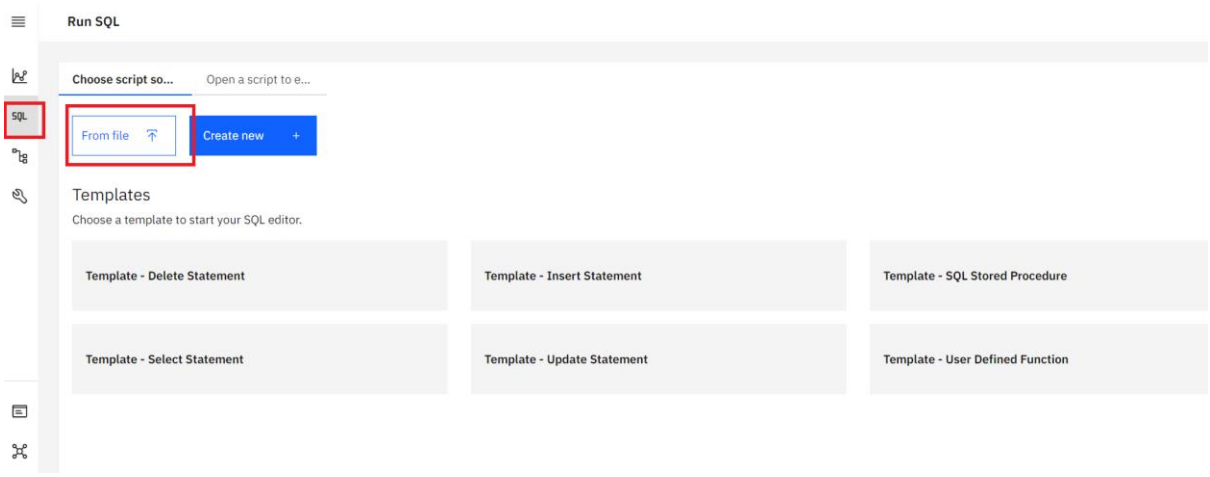
In this exercise, you will learn how to execute a script containing the CREATE TABLE commands for all the tables rather than create each table manually by typing the DDL commands in the SQL editor.

1. Download the script file to your computer:
   - HR_Database_Create_Tables_Script.sql

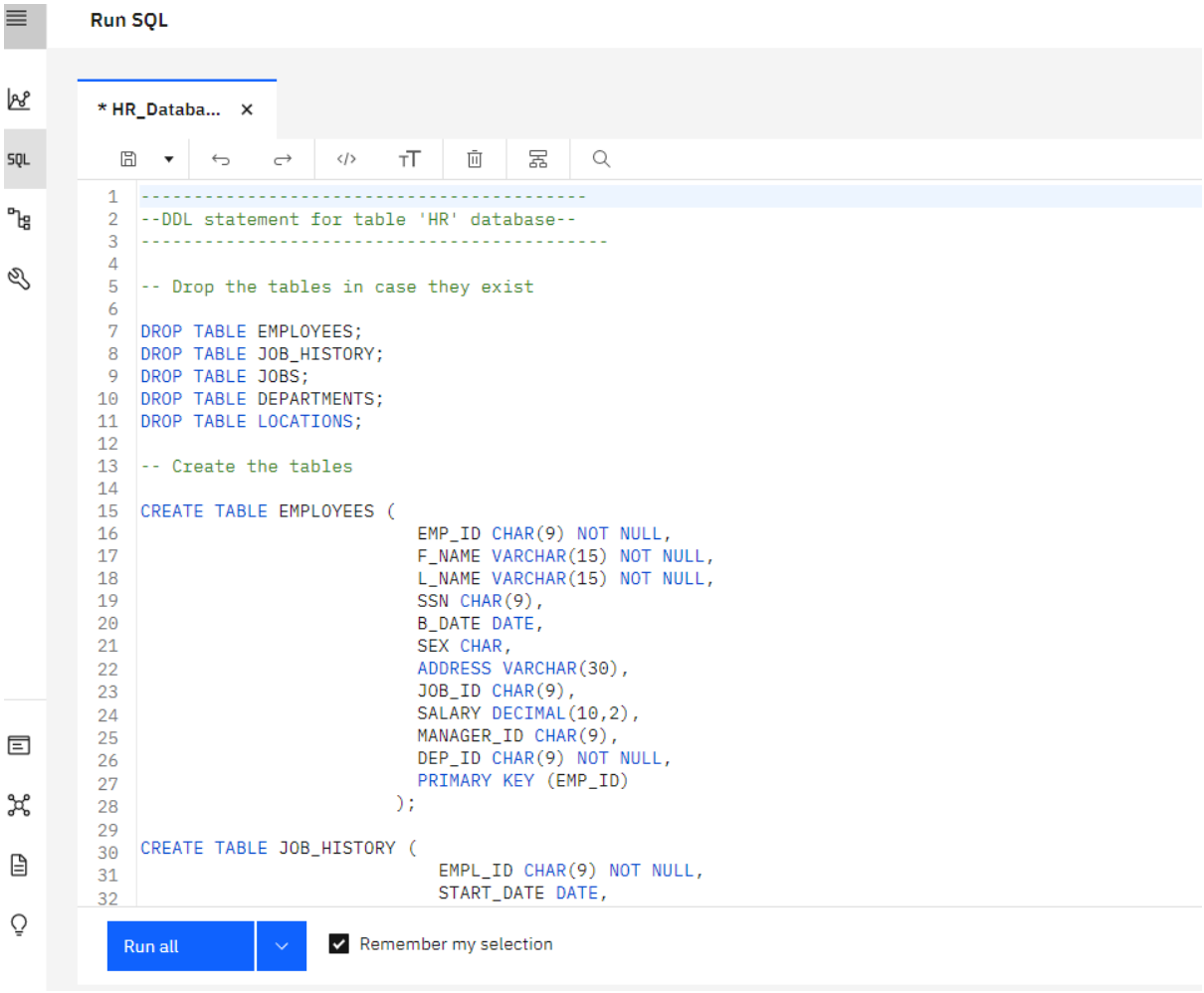
2. Login to IBM Cloud and go to the Resource List where you can find the Db2 service instance that you created in a previous lab under **Services** section. Click on the Db2-xx service. Next, click on **Go to UI** button.



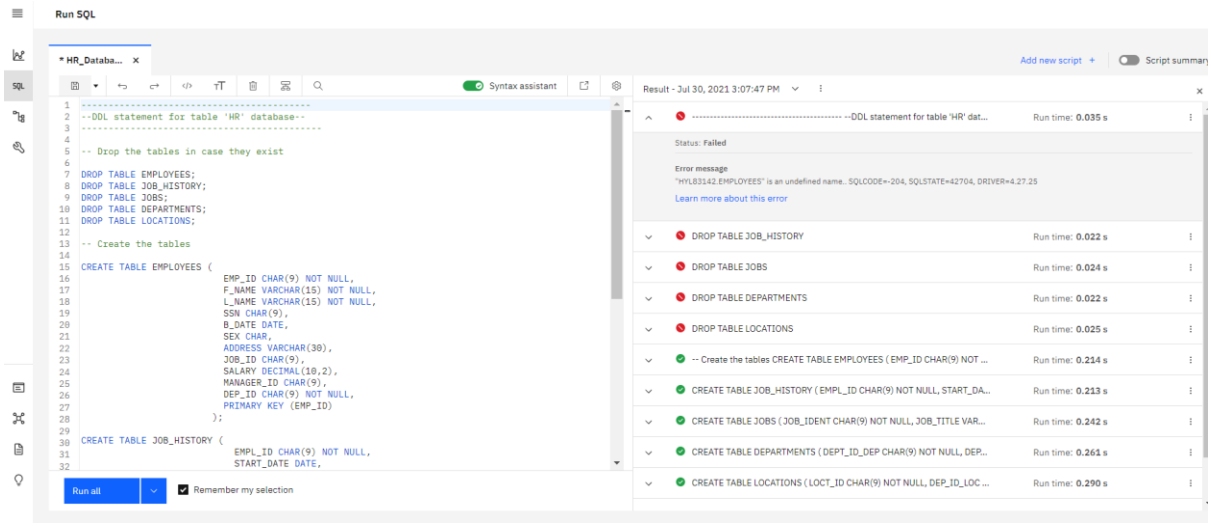
3. Click on Run SQL on the left corner and select the from file option.



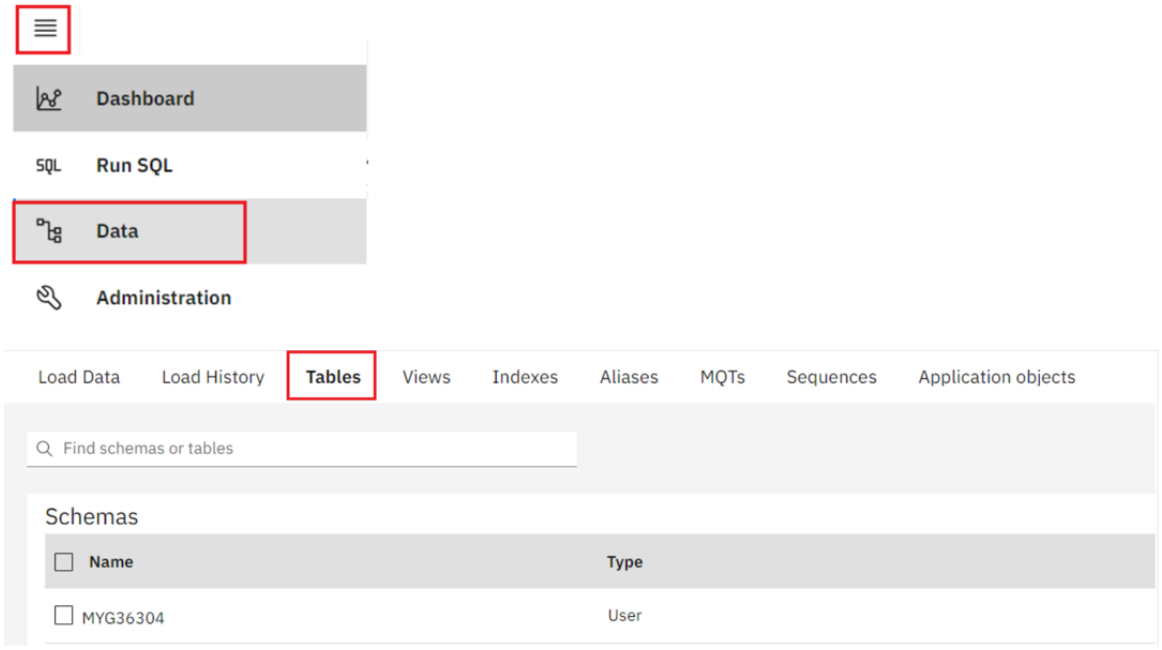
4. Locate the file **HR_Database_Create_Tables_Script.sql** that you downloaded to your computer earlier and open it.
5. Once the statements are in the SQL Editor tool , you can run the queries against the database by selecting the **Run All** button.



6. On the right side of the SQL editor window you will see a Result section. Clicking on a query in the Result section will show the execution details of the job like whether it ran successfully, or had any errors or warnings. Ensure your queries ran successfully and created all the tables.
    - **Note:** You may see several errors before the successful creation of the tables. These errors relate to the dropping (removal) of any pre-existing version of these tables. You can ignore these errors.

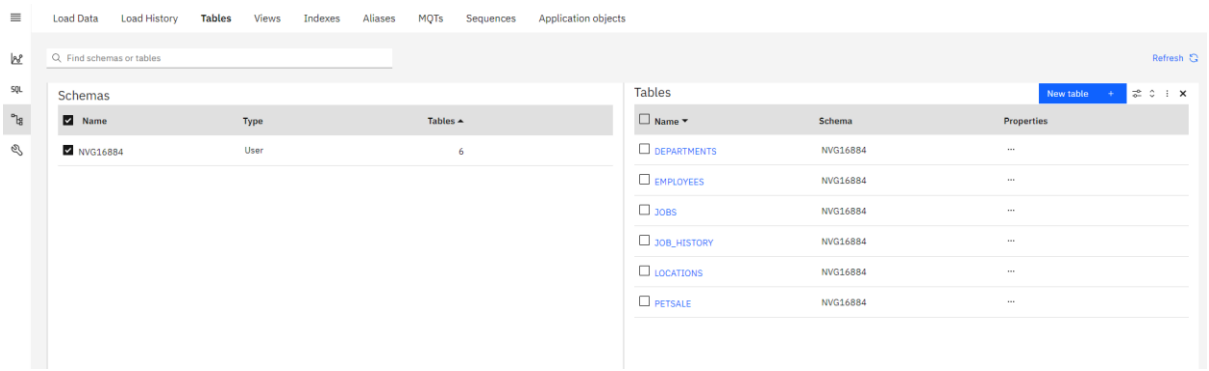


7. Now you can look at the tables you created. Click on the data icon and then click on Tables tab

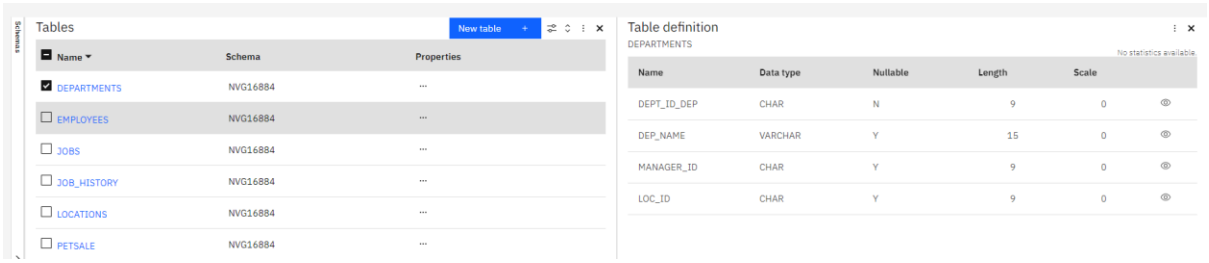


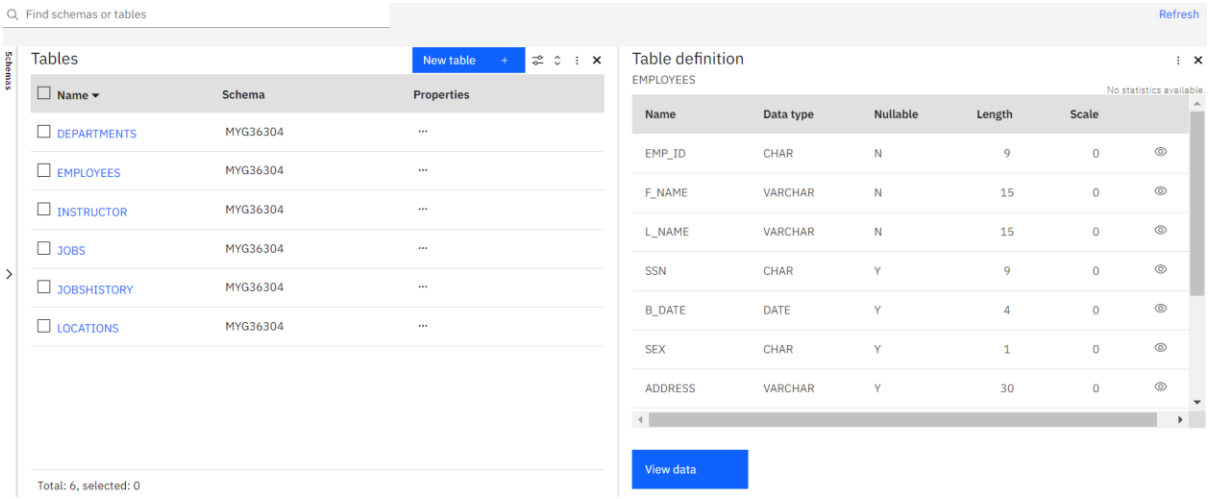
8. Select the Schema corresponding to your Db2 userid. It typically starts with 3 letters (not SQL) followed by 5 numbers (but will be different from the **MYG36304** example below). Then on the right side of the screen you should see the 5 newly created tables listed – DEPARTMENTS, EMPLOYEES, JOBS, JOB_HISTORY and LOCATIONS

(plus any other tables you may have created in previous labs e.g. PETSALE, PETRESCUE, etc.).



9. Click on any of the tables and you will see its Table Definition (that is, its list of columns, data types, etc).





Exercise 2: Load data into tables

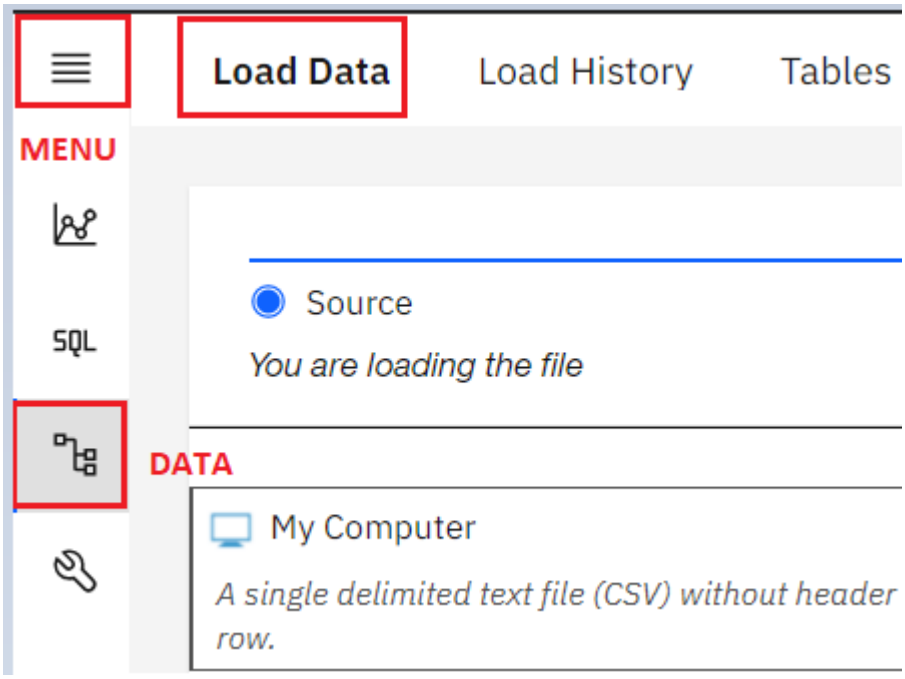
In this exercise, you will learn how data can be loaded into Db2. You could manually insert each row into the table one by one, but that would take a long time. Instead, Db2 (and almost every other database) allows you to load data from .CSV files.

The steps below explain the process of loading data into the tables you created earlier in exercise 1.

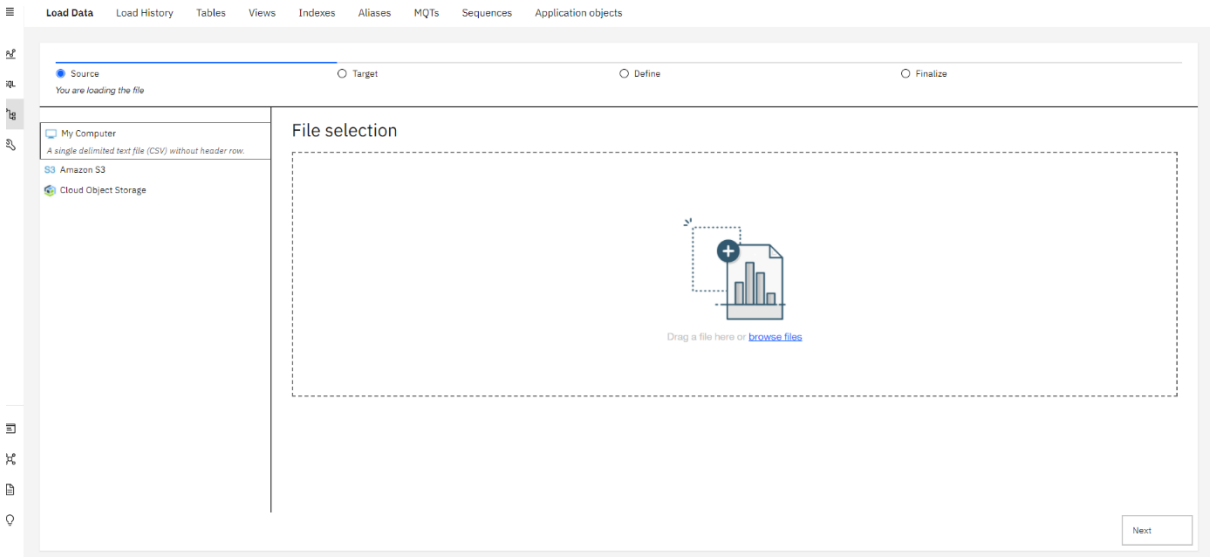
1. Download the 5 .csv files below to your local computer:
   - Departments.csv
   - Employees.csv

- Jobs.csv
- Locations.csv
- JobsHistory.csv

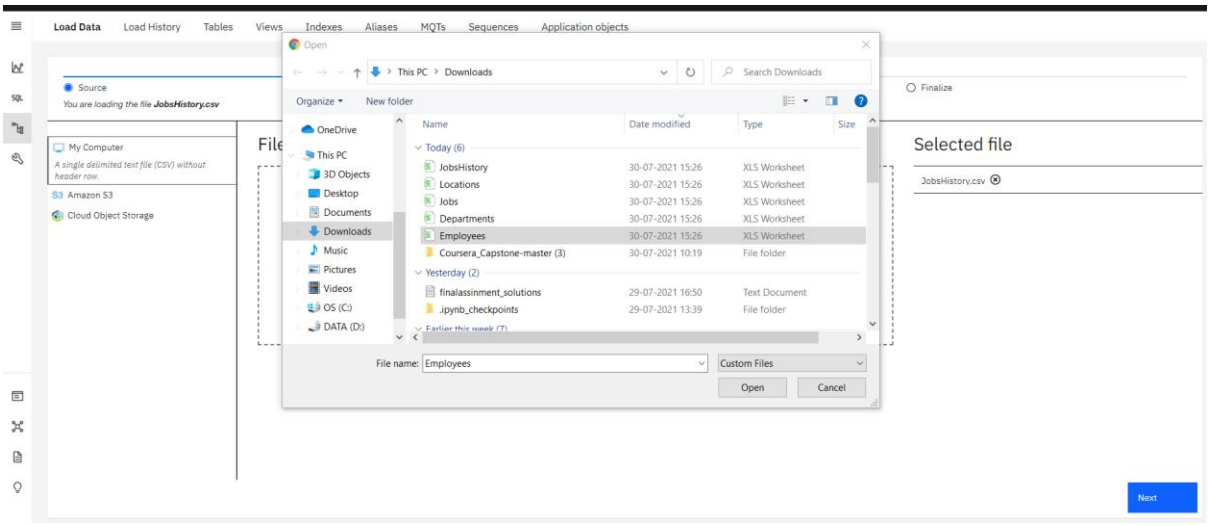
2. In the Db2 Console, from the 3-bar menu icon in the top left corner, click **Load**, and then select **Load Data**.



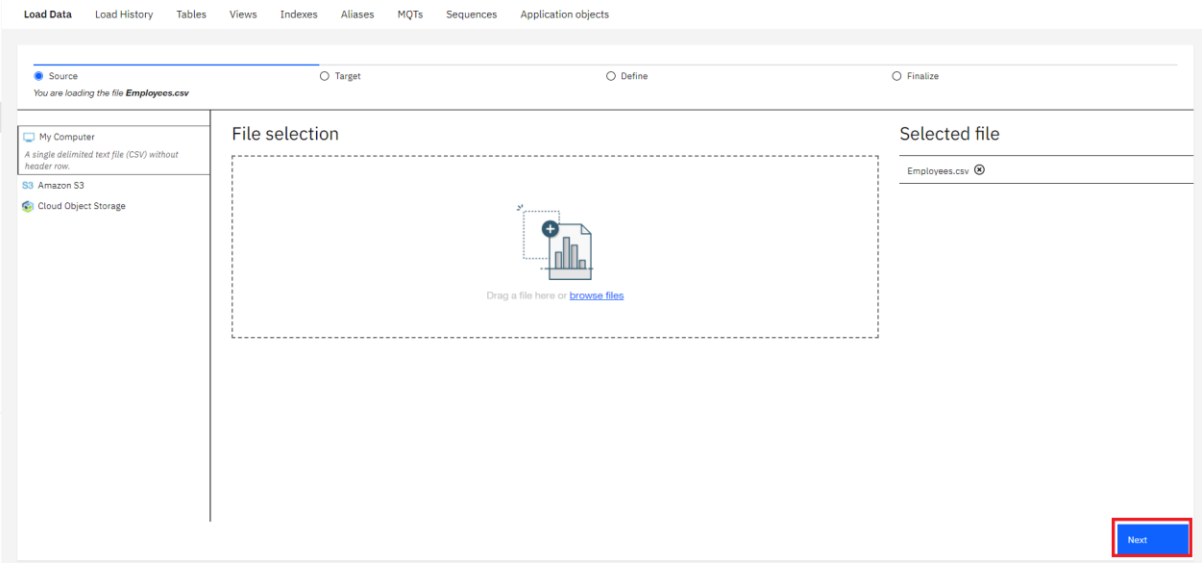
3. On the **Load Data** page that opens, ensure **My Computer** is selected as the source. Click on the **browse files** link.



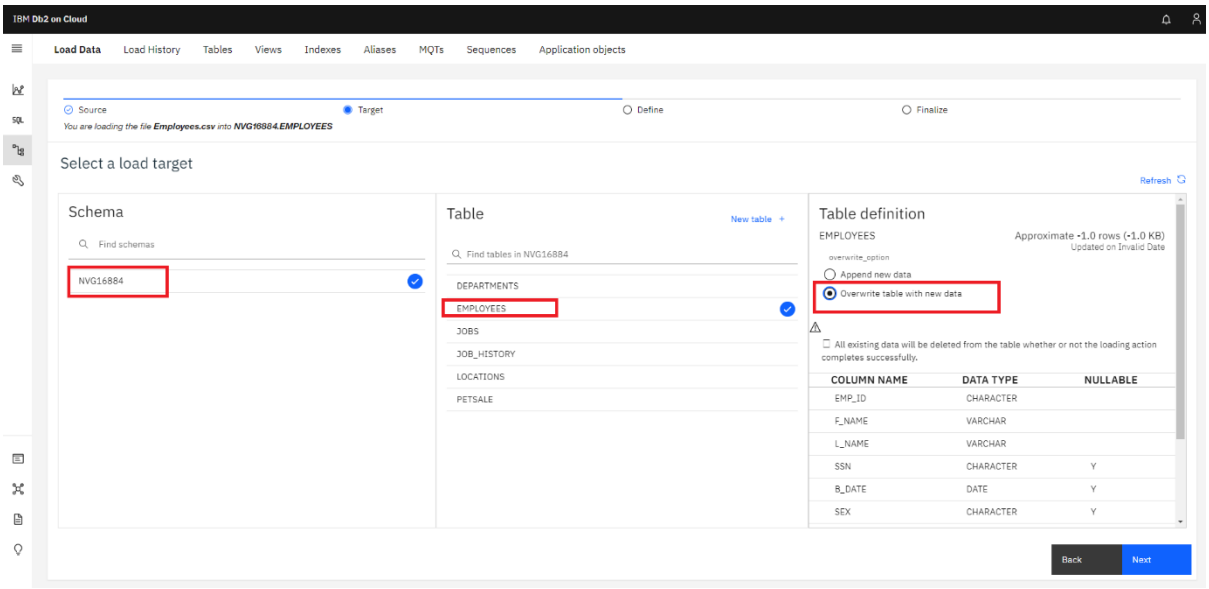
4. Choose the file **Employees.csv** that you downloaded to your computer and click **Open**.



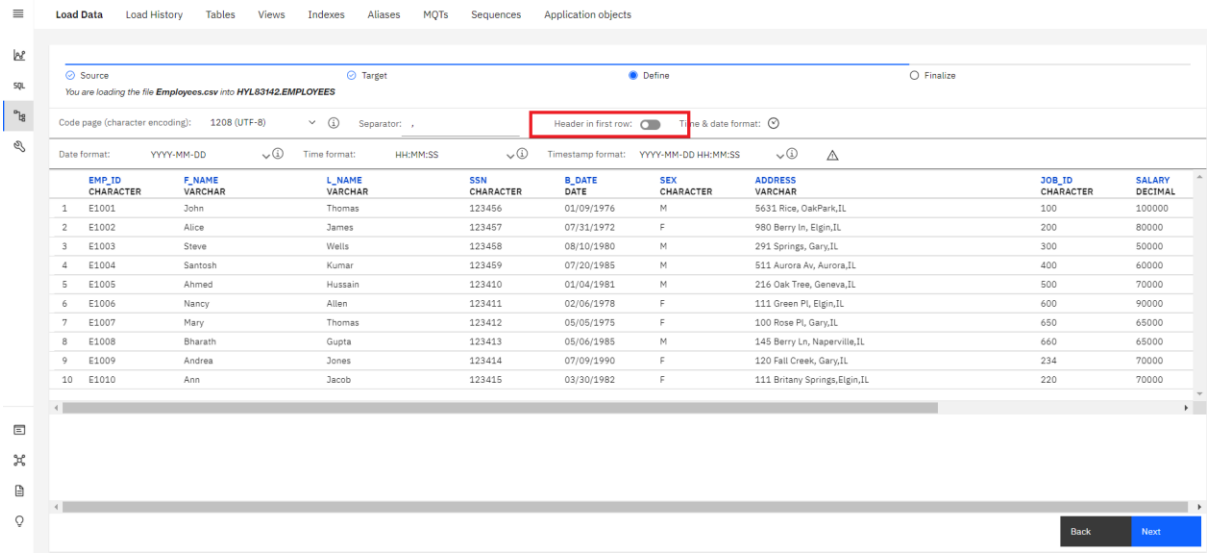
5. Once the File is selected, click **Next** in the bottom right corner.



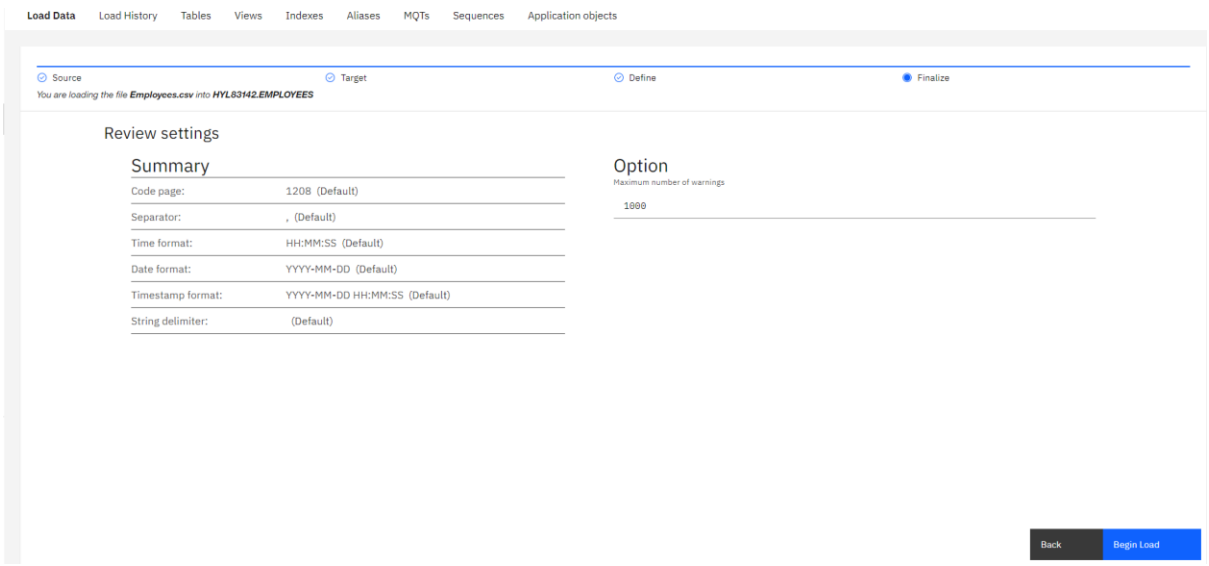
6. Select the schema for your Db2 Userid (the one where you created the tables earlier).It will show all the tables that have been created in this schema previously, including the Employees table. Select the **EMPLOYEES** table, and in the new Table Definition tab that appears, choose **Overwrite table with new data** (note the warning message), then click **Next**. Select the **Employees** table.



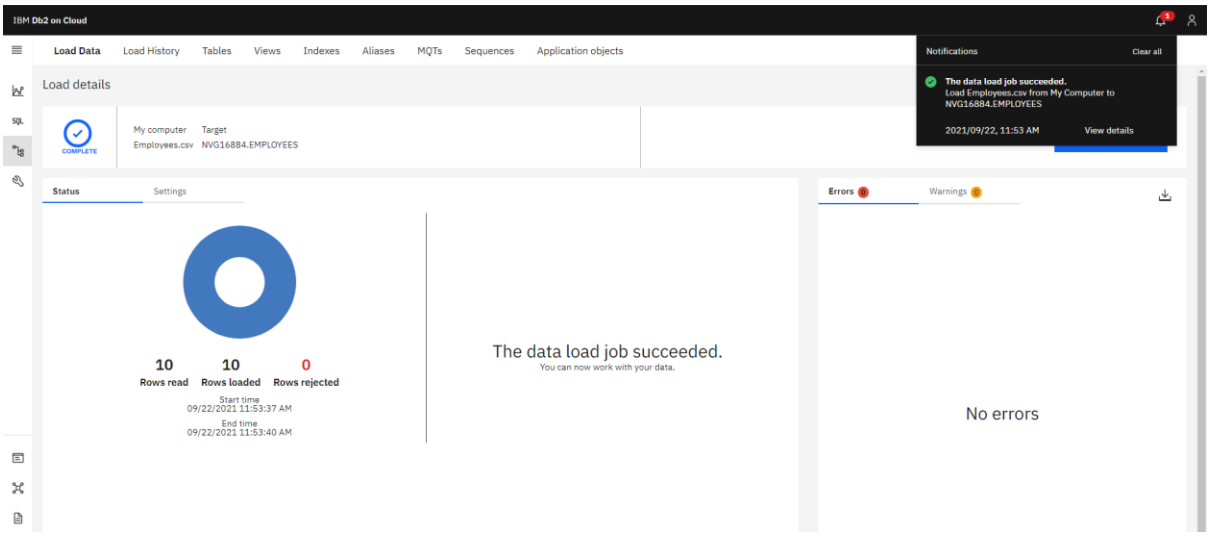
7. Since the source data files do not contain any rows with column labels, **turn off** the setting for **Header in first row**. Also, click on the down arrow next to **Date format** and choose **MM/DD/YYYY** since that is how the date is formatted in the source file.



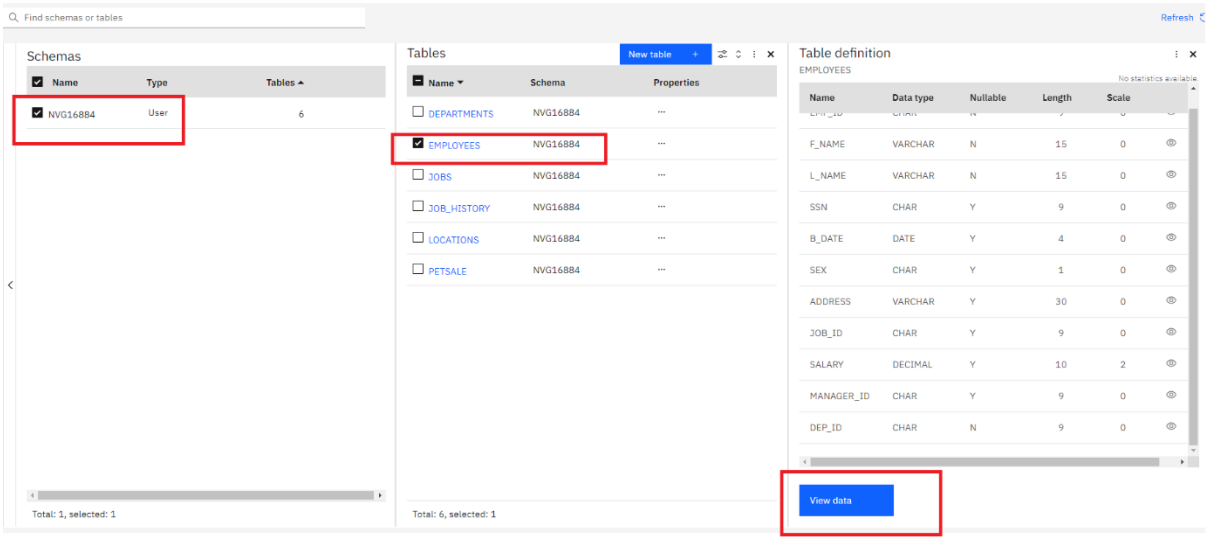
8. Click **Next**. Review the load settings and click **Begin Load** in the bottom right corner.



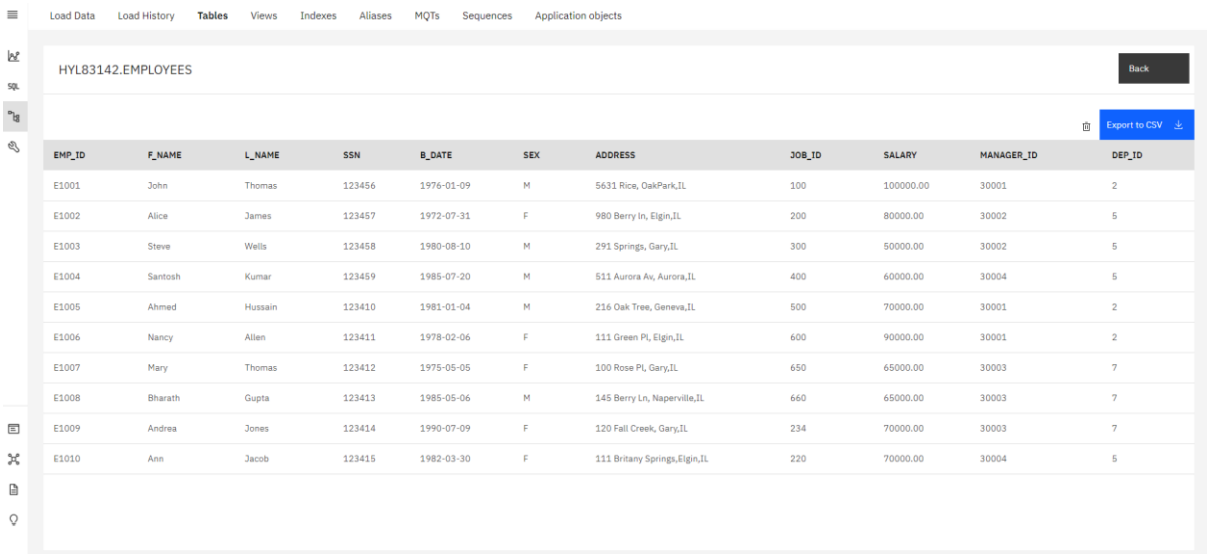
9. After loading has completed, you will notice that you were successful in loading all 10 rows of the Employees table. If there are any **Errors** or **Warnings**, you can see them on this screen.



10. Click on the **Tables** tab and then select the **EMPLOYEES** table and then click on **View data**.



11. Now you can view the table data.



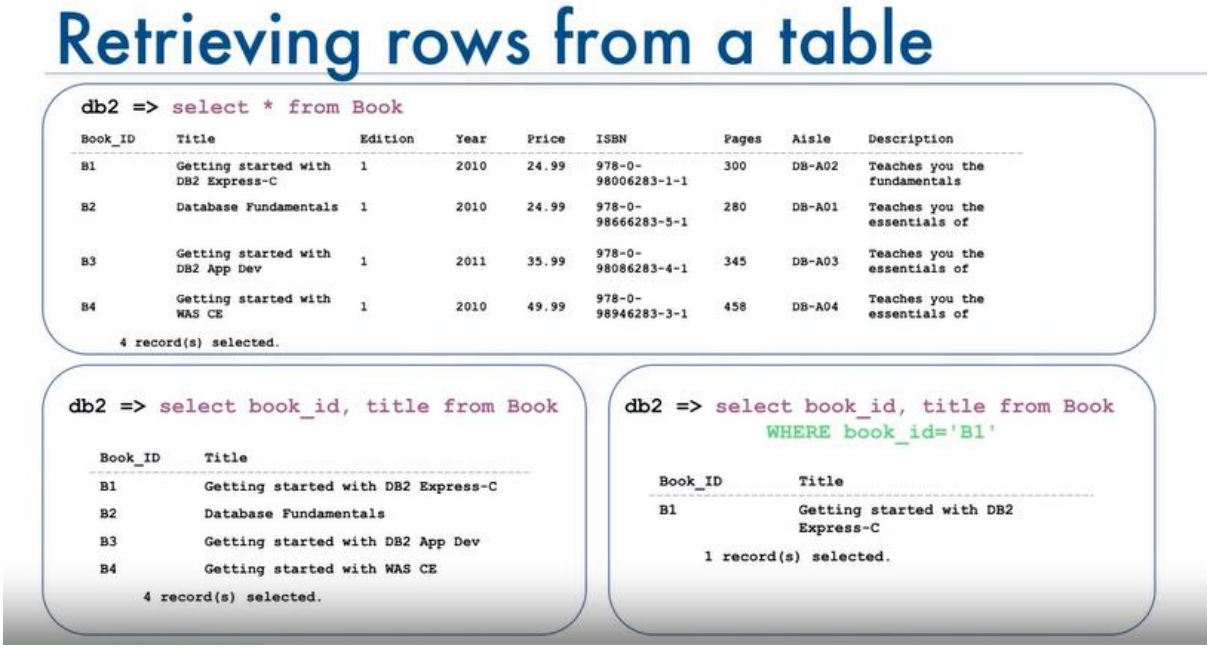
12. Now it's your turn to load data to the remaining 4 tables of the HR database – **LOCATIONS**, **JOB_HISTORY**, **JOBS**, and **DEPARTMENTS** from the remaining source files.
13. Click **Load More Data** and then follow the steps from **Step 3** above again to load the remaining 4 tables.

IMPORTANT Make sure you perform the steps in **Step 7** for each of the 4 remaining file loads.

Using String Patterns and Ranges

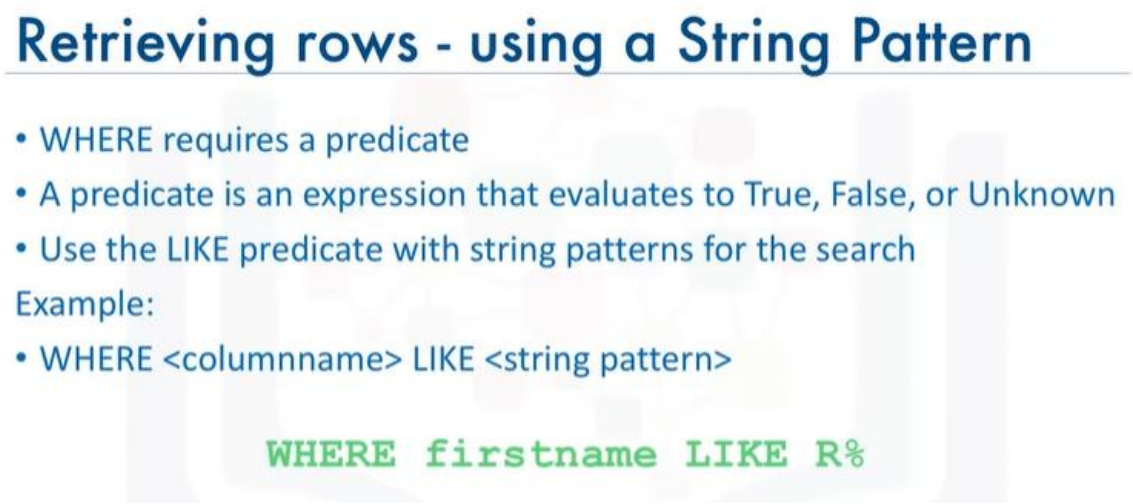
- At the end of this lesson, you will be able to describe how to simplify a SELECT statement by using:
  - String patterns
  - Ranges, or
  - Sets of values

The main purpose of a database management system is not just to store the data, but also facilitate retrieval of the data. In its simplest form, a SELECT statement is select star from table name. Based on a simplified library database model and the table Book, SELECT star from Book gives a result set of four rows.



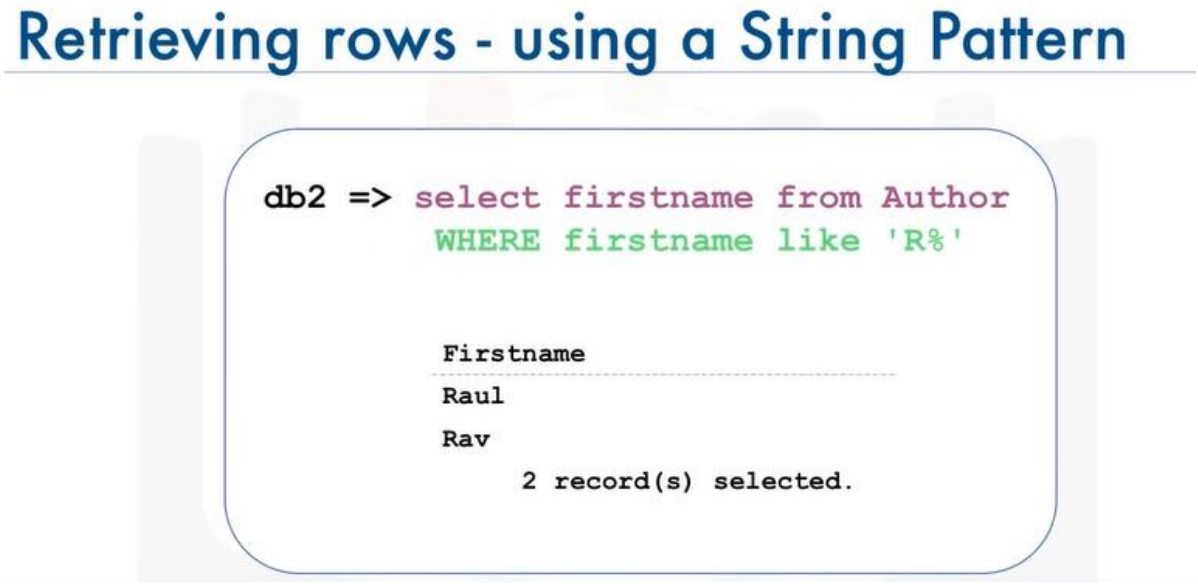
All the data rows for all columns in the table Book are displayed or you can retrieve a subset of columns for example, just two columns from the table book such as Book_ID and Title. Or you can restrict the result set by using the WHERE clause. For example, you can select the title of the book whose Book_ID is B1.

But what if we don't know exactly what value to specify in the WHERE clause?

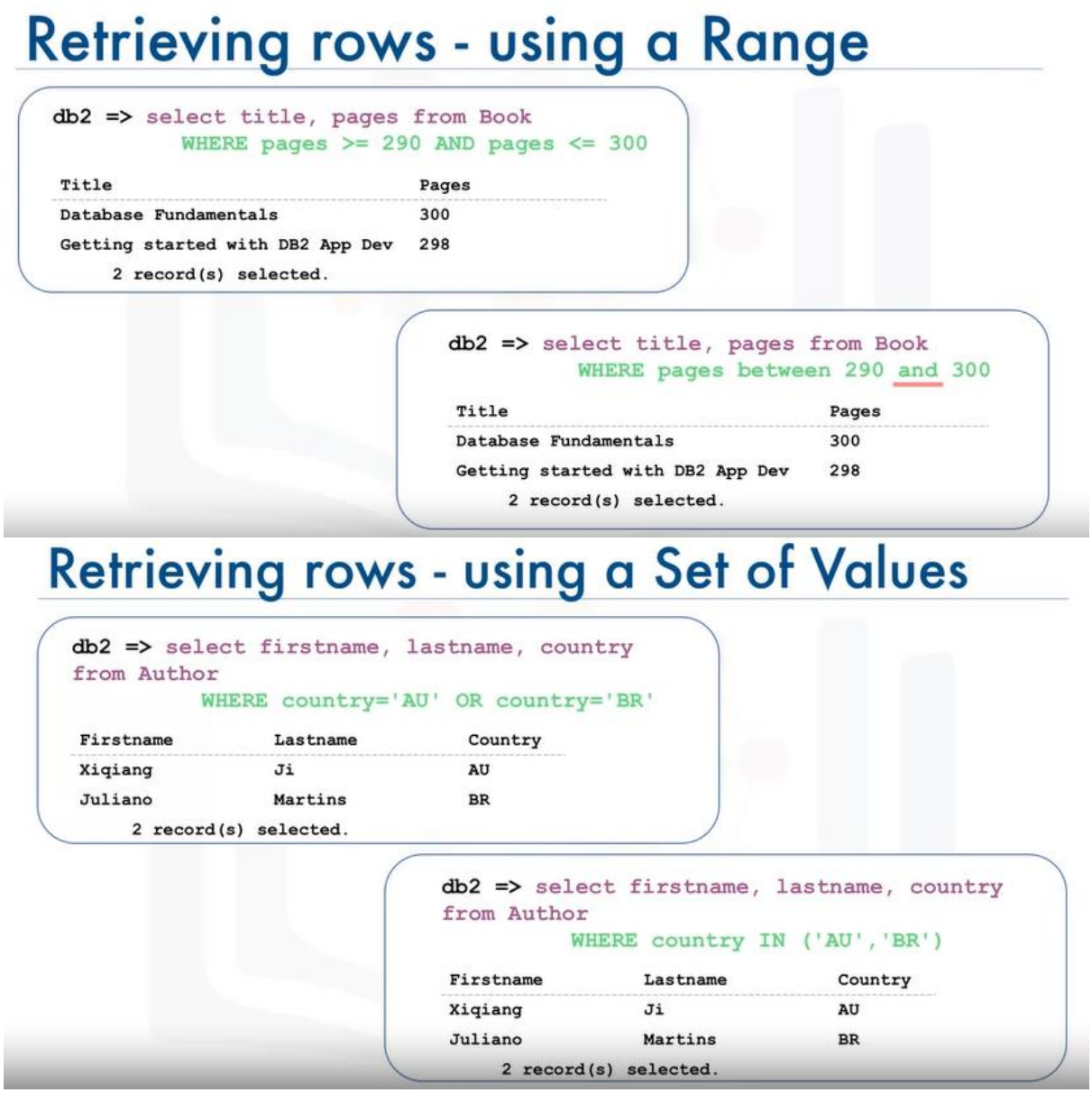


- The WHERE clause always requires a predicate, which is a condition that evaluates to true, false, or unknown. But what if we don't know exactly what value the predicate is? For example, what if we can't remember the name of the author, but we remember that their first name starts with R?
- In a relational database, we can use string patterns to search data rows that match this condition. Let's look at some examples of using string patterns. If we can't remember the name of the author, but we remember that their name starts with R, we use the WHERE clause with the like predicate
- The like predicate is used in a WHERE clause to search for a pattern in a column. The percent sign is used to define missing letters. The percent sign can be placed before the pattern, after the pattern, or both before and after the pattern.
- In this example, we use the percent sign after the pattern, which is the letter R. The percent sign is called a wildcard character. A wildcard character is used to substitute other characters.

➢ So, if we can't remember the name of the author, but we can remember that their first name starts with the letter R, we add the like predicate to the WHERE clause.



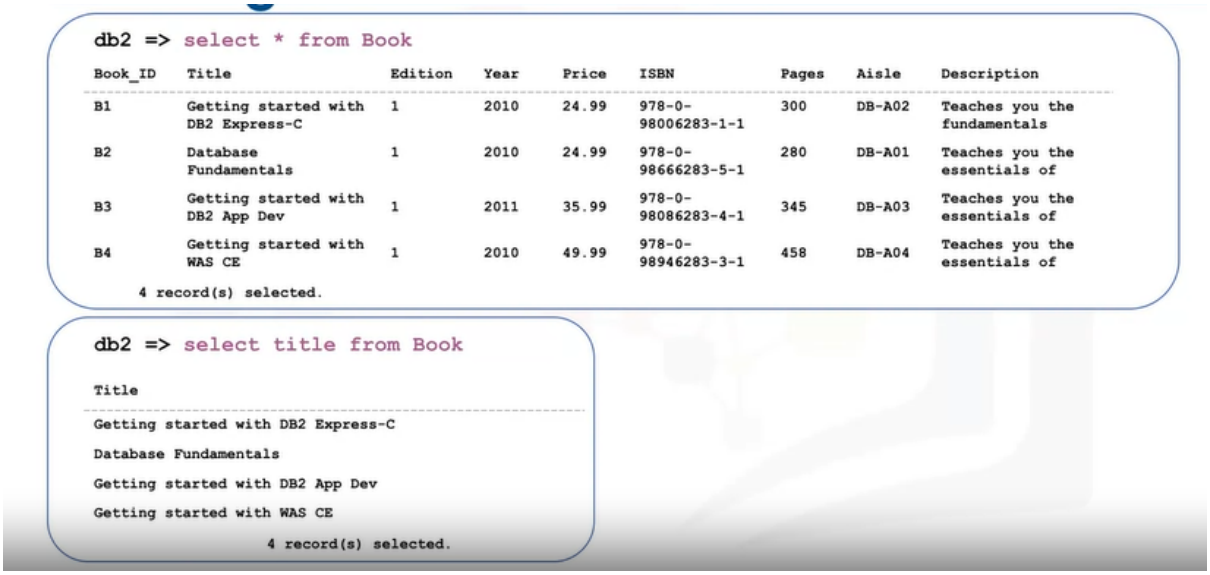
➢ For example, select first name from author, where firstname like 'R%'. This will return all rows in the author table whose author's first name starts with the letter R. And here is the result set. Two rows a return for authors Raul and Rav.



Sorting Result Sets

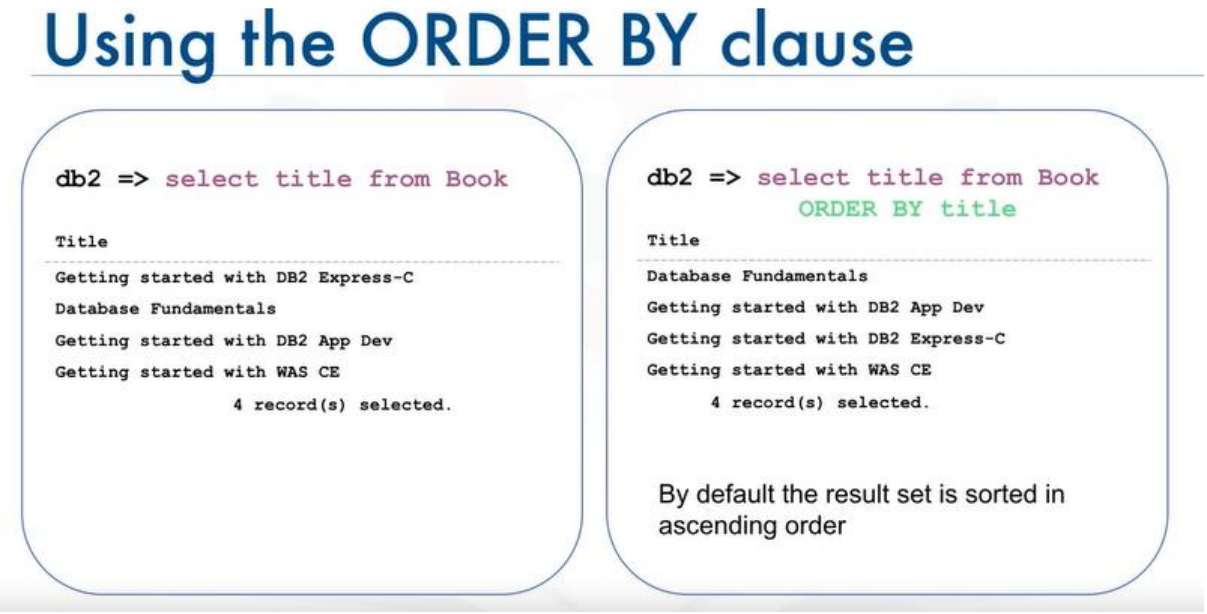
- At the end of this lesson, you will be able to:
  - Describe how to sort the result set by either ascending or descending order
  - Explain how to indicate which column to use for the sorting order

The main purpose of a database management system is not just to store the data, but also facilitate retrieval of the data.



n its simplest form, a select statement is select * from table name. Based on our simplified library database model, in the table book, select * from book gives a result set of four rows. All the data rows for all columns in the table book are displayed. We can choose to list the book titles only as shown in this example, select title from book. However, the order does not seem to be in any order.

Displaying the results set in alphabetical order would make the result set more convenient. To do this, we use the "order by" clause.



The order by clause is used in a query to sort the result set by a specified column. In this example, we have used order by on the column title to sort the result set. By default, the result set is sorted in ascending order. In this example, the result set is sorted in alphabetical order by book title.

ORDER BY clause – Descending order

```
db2 => select title from Book
          ORDER BY title

Title
----------------------------------------
Database Fundamentals
Getting started with DB2 App Dev
Getting started with DB2 Express-C
Getting started with WAS CE
      4 record(s) selected.
```

Ascending order by default

```
db2 => select title from Book
          ORDER BY title DESC

Title
----------------------------------------
Getting started with WAS CE
Getting started with DB2 Express-C
Getting started with App Dev
Database Fundamentals
      4 record(s) selected.
```

Descending order with DESC keyword

Specifying Column Sequence Number

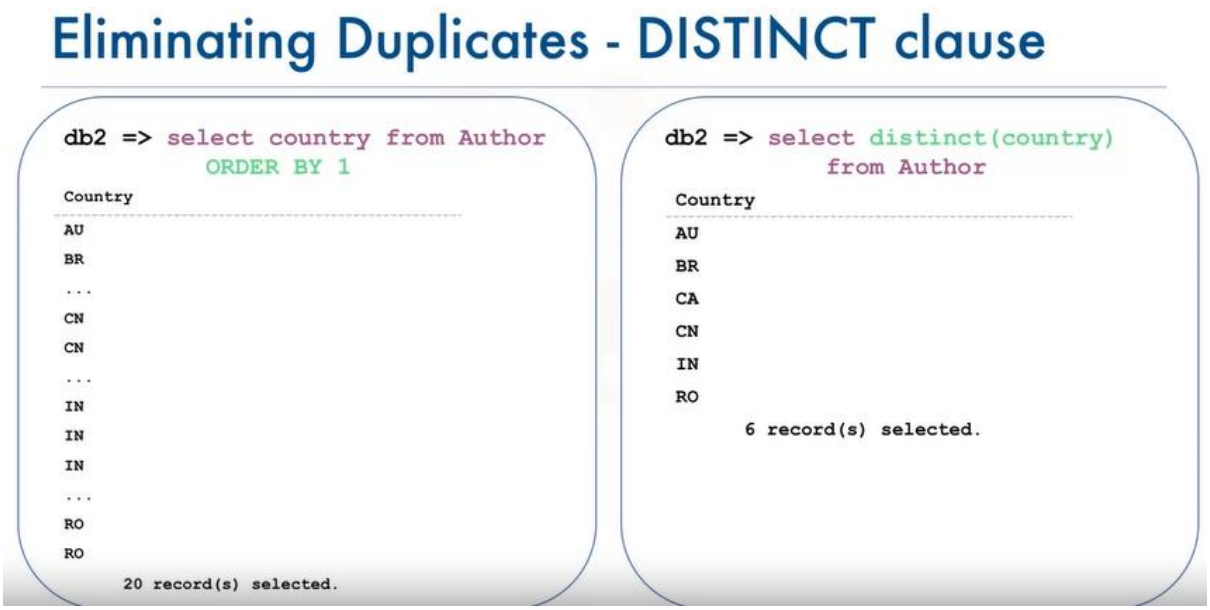
```
db2 => select title, pages from Book
          ORDER BY 2

Title                                    Pages
---------------------------------------- -----
Getting started with WAS CE              278
Getting started with DB2 Express-C       280
Getting started with App Dev             298
Database Fundamentals                    300
      4 record(s) selected.
```

Ascending order by Column 2 (number of pages)

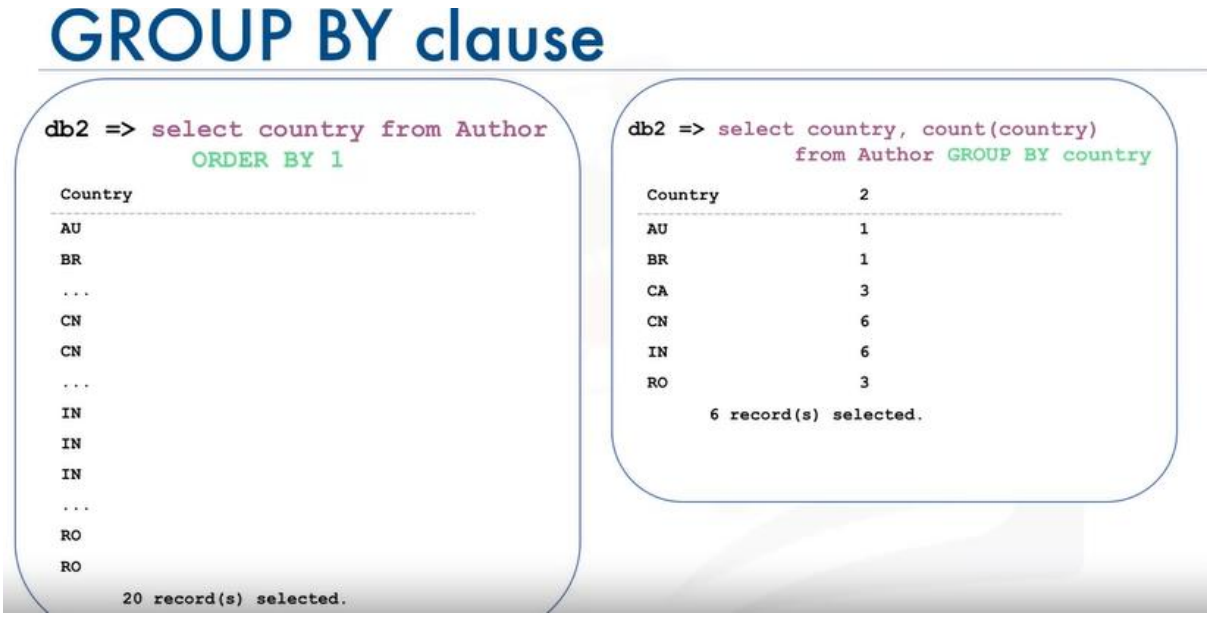
Grouping Result Sets

Grouping Result Sets

- At the end of this lesson, you will be able to:
  - Eliminate duplicates from a result set
  - Describe how to further restrict a result set

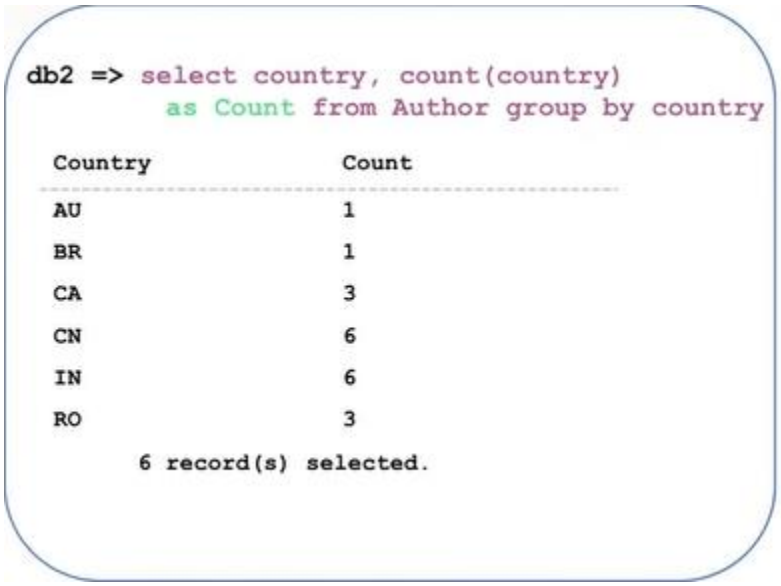


At times, a select statement result set can contain duplicate values. Based on our simplified library database model, in the author table example, the country column lists the two-letter country code of the author's country. If we select just the country column, we get a list of all of the countries. For example, select country from author order by 1. The order by clause sorts the result set. This result set lists the countries the authors belong to, sorted alphabetically by country. In this case, the result set displays 20 rows, one row for each of the 20 authors. But some of the authors come from the same country, so the result set contains duplicates. However, all we need is a list of countries the authors come from. So in this case, duplicates do not make sense. To eliminate duplicates, we use the keyword distinct. Using the keyword "distinct" reduces the result set to just six rows.

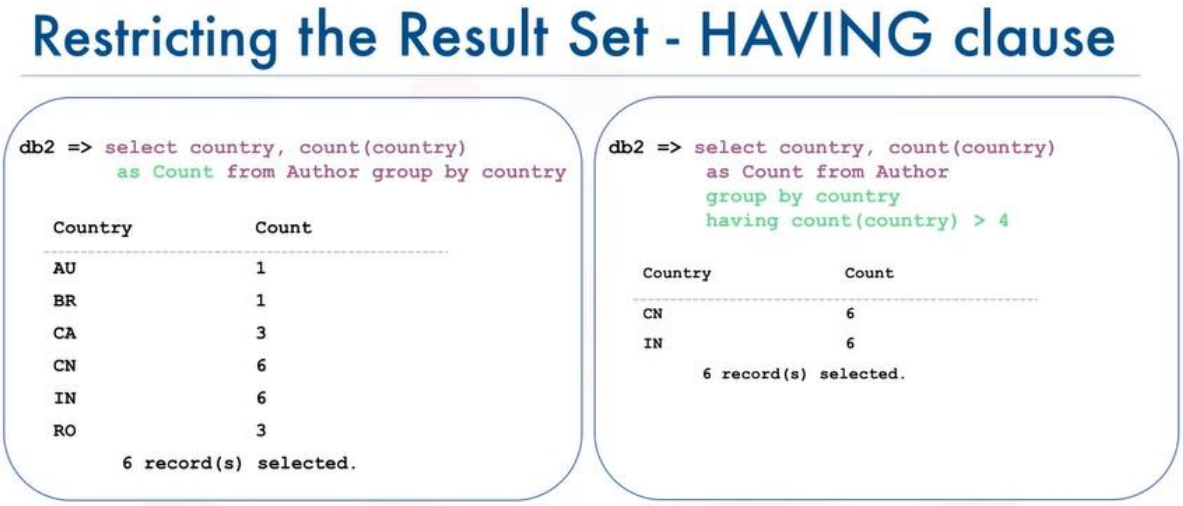


But what if we wanted to also know how many authors come from the same country? So now we know that the 20 authors come from six different countries. But we might want to also know how many authors come from the same country. To display the result set listing the country and number of authors that come from that country, we add the "group by" clause to the select

statement. The "group by" clause groups a result into subsets that has matching values for one or more columns. In this example, countries are grouped and then counted using the count function.



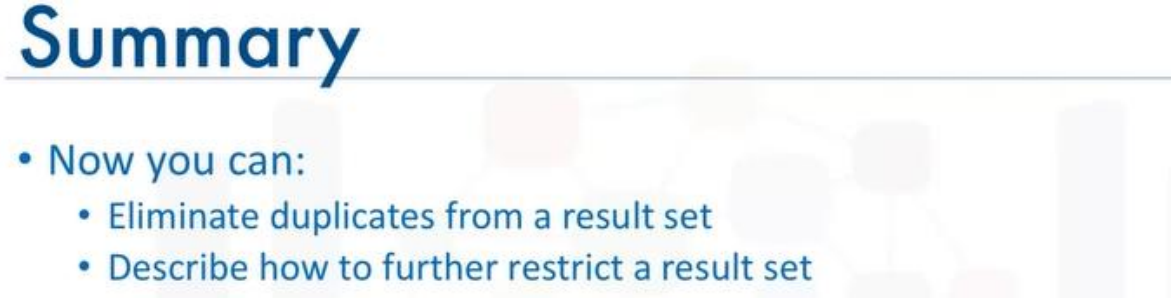
```
db2 => select country, count(country)
         as Count from Author group by country

Country                 Count
-----------------------------------------------
AU                      1
BR                      1
CA                      3
CN                      6
IN                      6
RO                      3

       6 record(s) selected.
```



Restricting the Result Set - HAVING clause

```
db2 => select country, count(country)
         as Count from Author group by country

Country                 Count
-----------------------------------------------
AU                      1
BR                      1
CA                      3
CN                      6
IN                      6
RO                      3

       6 record(s) selected.
```

```
db2 => select country, count(country)
         as Count from Author
         group by country
         having count(country) > 4

Country                 Count
-----------------------------------------------
CN                      6
IN                      6

       6 record(s) selected.
```



Summary

- Now you can:
  - Eliminate duplicates from a result set
  - Describe how to further restrict a result set

Hands-on Lab: String Patterns, Sorting and Grouping

Exercise 1: String Patterns

In this exercise, you will go through some SQL problems on String Patterns.

1. Problem:

   Retrieve all employees whose address is in Elgin,IL.

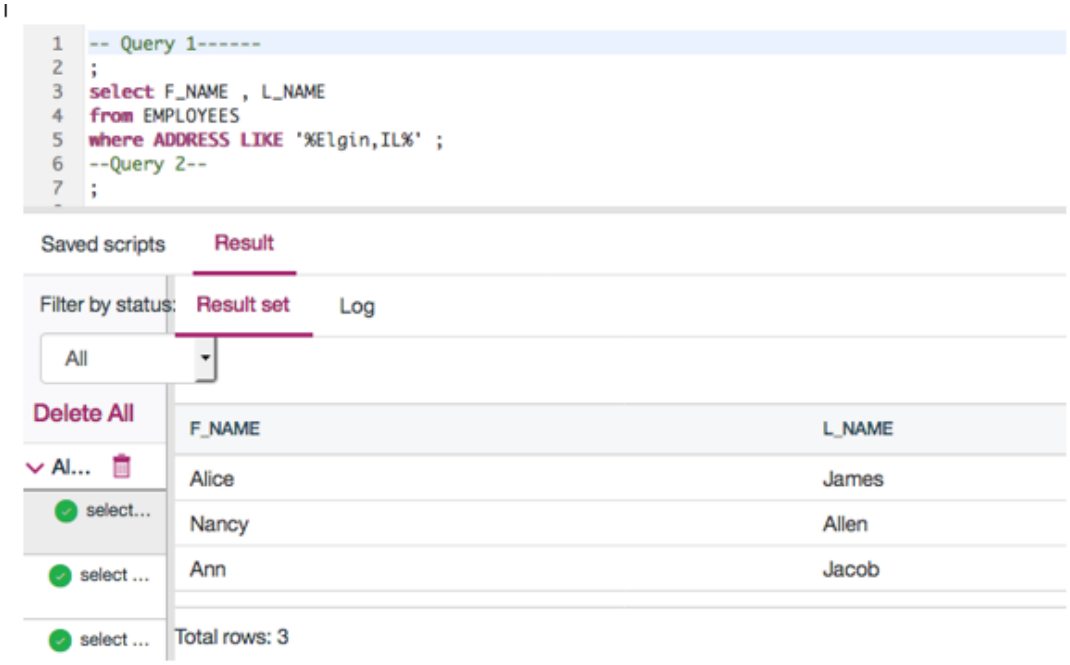
Hint

Use the LIKE operator to find similar strings.

Solution

```
SELECT F_NAME , L_NAME
FROM EMPLOYEES
WHERE ADDRESS  LIKE '%Elgin,IL%';
```

Output



2. Problem:

Retrieve all employees who were born during the 1970's.

Hint

Use the LIKE operator to find similar strings.

Solution

```
SELECT F_NAME , L_NAME
FROM EMPLOYEES
WHERE B_DATE  LIKE '197%';
```

Output

```
--Query 2--
;
select F_NAME , L_NAME
from EMPLOYEES
where B_DATE LIKE '197%' ;
---Query3--
:
```

Saved scripts | Result

Filter by status: | Result set | Log

All

Delete All

F_NAME	L_NAME
John	Thomas
Alice	James
Nancy	Allen
Mary	Thomas

Total rows: 4

3. Problem:

Retrieve all employees in department 5 whose salary is between 60000 and 70000.

Hint

Use the keyword BETWEEN for this SQL problem.

Solution

```
SELECT *
FROM EMPLOYEES
WHERE (SALARY BETWEEN 60000 AND 70000) AND DEP_ID = 5;
```

Output

```
---Query3--
;
select *
from EMPLOYEES
where (SALARY BETWEEN 60000 and 70000)  and DEP_ID = 5 ;
--Query4--
;
```

Saved scripts | Result

Filter by status: | Result set | Log

All

Delete All

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1004	Santosh	Kumar	1234...	1985-07-20	M	511 Aurora ...	400	60000.00	30004	5
E1010	Ann	Jacob	12341...	1982-03-30	F	111 Britany ...	220	70000.00	30004	5

Total rows: 2

Exercise 2: Sorting

In this exercise, you will go through some SQL problems on Sorting.

1. Problem:

   Retrieve a list of employees ordered by department ID.

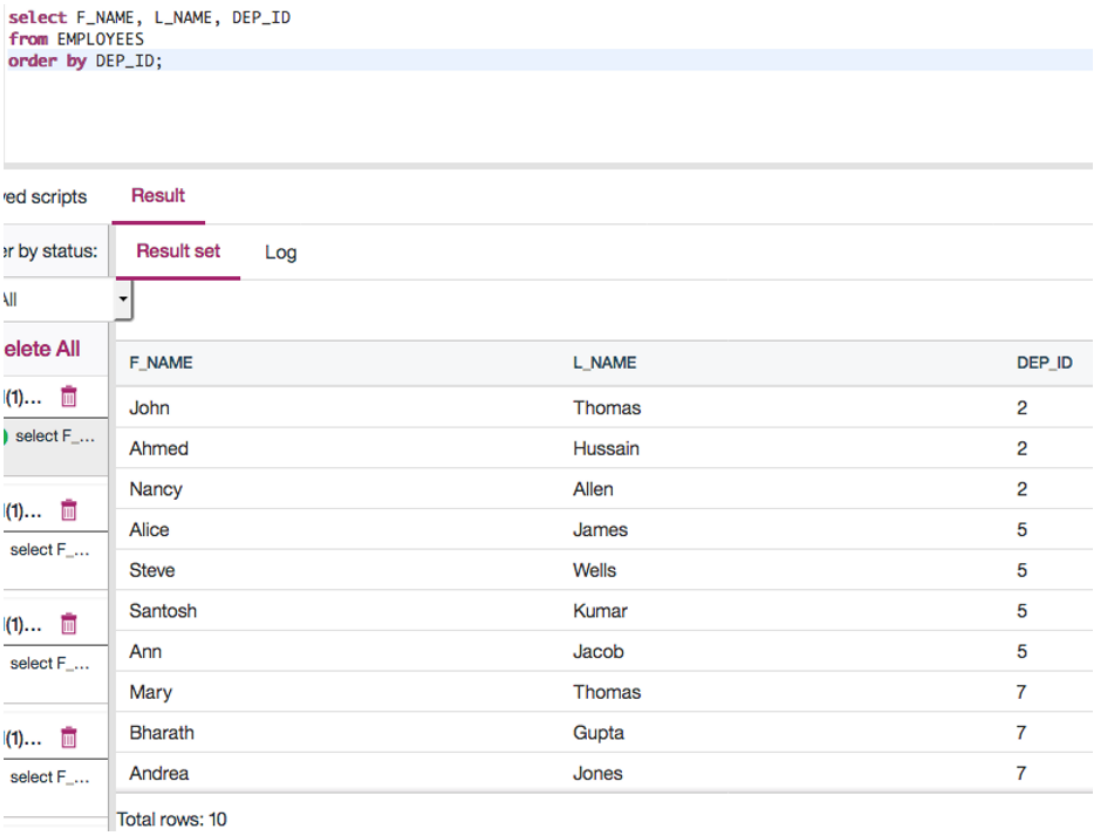
   Hint

   Use the ORDER BY clause for this SQL problem. By default, the ORDER BY clause sorts the records in ascending order.

   Solution

```
SELECT F_NAME, L_NAME, DEP_ID
FROM EMPLOYEES
ORDER BY DEP_ID;
```

   Output



2. Problem:

   Retrieve a list of employees ordered in descending order by department ID and within each department ordered alphabetically in descending order by last name.

   Solution

```
SELECT F_NAME, L_NAME, DEP_ID
FROM EMPLOYEES
ORDER BY DEP_ID DESC, L_NAME DESC;
```

   Output

```
select F_NAME, L_NAME, DEP_ID
from EMPLOYEES
order by DEP_ID desc, L_NAME desc;
```

Saved scripts | Result

Filter by status: | Result set | Log

All

Delete All

All(1)... select F_...

All(1)... select F_...

All(1)... select F_...

All(1)... select F_...

All(1)...

F_NAME	L_NAME	DEP_ID
Mary	Thomas	7
Andrea	Jones	7
Bharath	Gupta	7
Steve	Wells	5
Santosh	Kumar	5
Alice	James	5
Ann	Jacob	5
John	Thomas	2
Ahmed	Hussain	2
Nancy	Allen	2

Total rows: 10

3. (Optional) Problem:

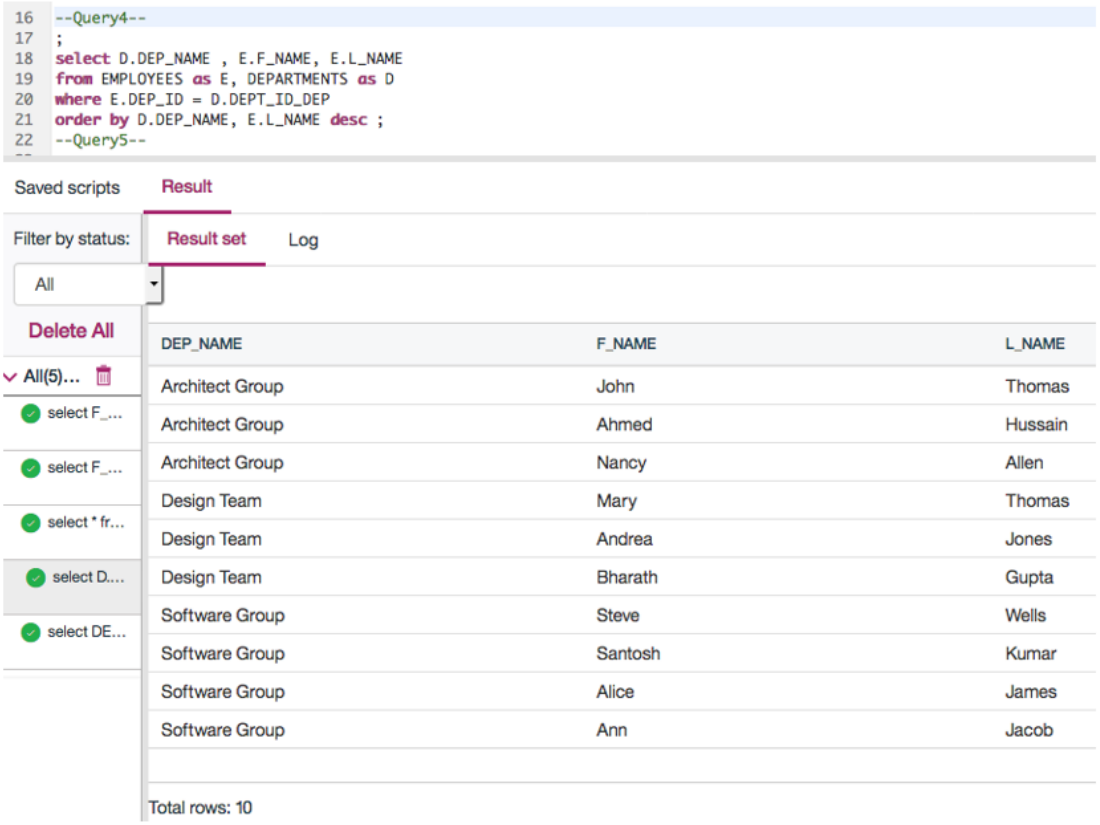
   In SQL problem 2 (Exercise 2 Problem 2), use department name instead of department ID. Retrieve a list of employees ordered by department name, and within each department ordered alphabetically in descending order by last name.

   Solution

```
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;
```

In the SQL Query above, D and E are aliases for the table names. Once you define an alias like D in your query, you can simply write D.COLUMN_NAME rather than the full form DEPARTMENTS.COLUMN_NAME.

Output



Exercise 3: Grouping

In this exercise, you will go through some SQL problems on Grouping.

NOTE: The SQL problems in this exercise involve usage of SQL Aggregate functions AVG and COUNT. COUNT has been covered earlier. AVG is a function that can be used to calculate the Average or Mean of all values of a specified column in the result set. For example, to retrieve the average salary for all employees in the EMPLOYEES table, issue the query: SELECT AVG(SALARY) FROM EMPLOYEES;. You will learn more about AVG and other aggregate functions later in the lecture **Built-in Database Functions**.

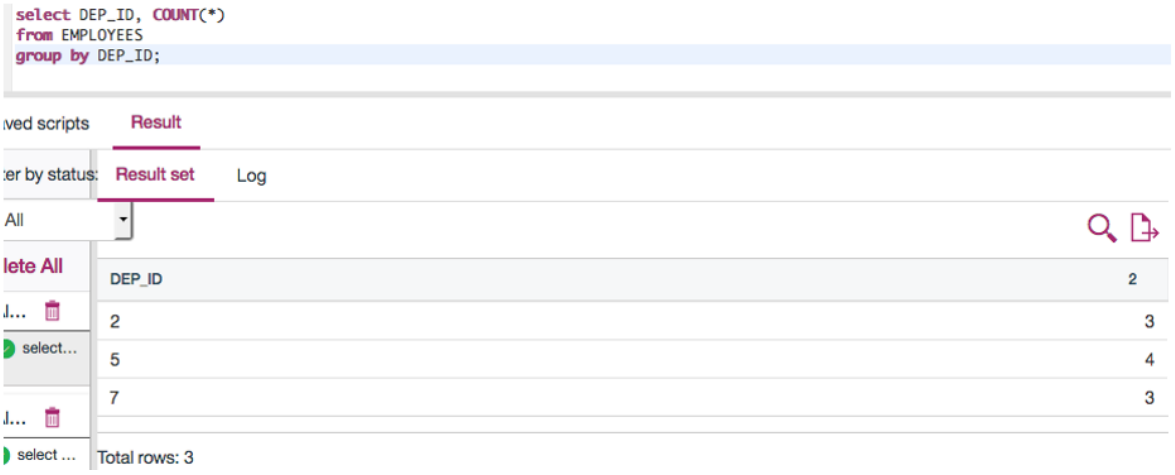
1. Problem:

   For each department ID retrieve the number of employees in the department.

Solution

```
SELECT DEP_ID, COUNT(*)
FROM EMPLOYEES
GROUP BY DEP_ID;
```

Output



2. Problem:

For each department retrieve the number of employees in the department, and the average employee salary in the department..

Solution

```
SELECT DEP_ID, COUNT(*), AVG(SALARY)
FROM EMPLOYEES
GROUP BY DEP_ID;
```

Output

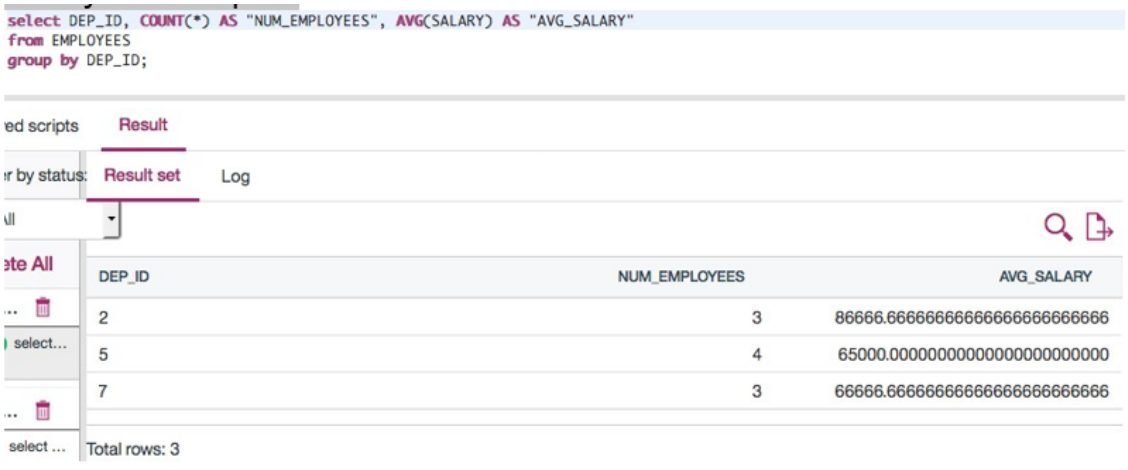
3. Problem:

   Label the computed columns in the result set of SQL problem 2 (Exercise 3 Problem 2) as NUM_EMPLOYEES and AVG_SALARY.

   Solution

```
SELECT DEP_ID, COUNT(*) AS "NUM_EMPLOYEES", AVG(SALARY) AS "AVG_SALARY"
FROM EMPLOYEES
GROUP BY DEP_ID;
```

   Output



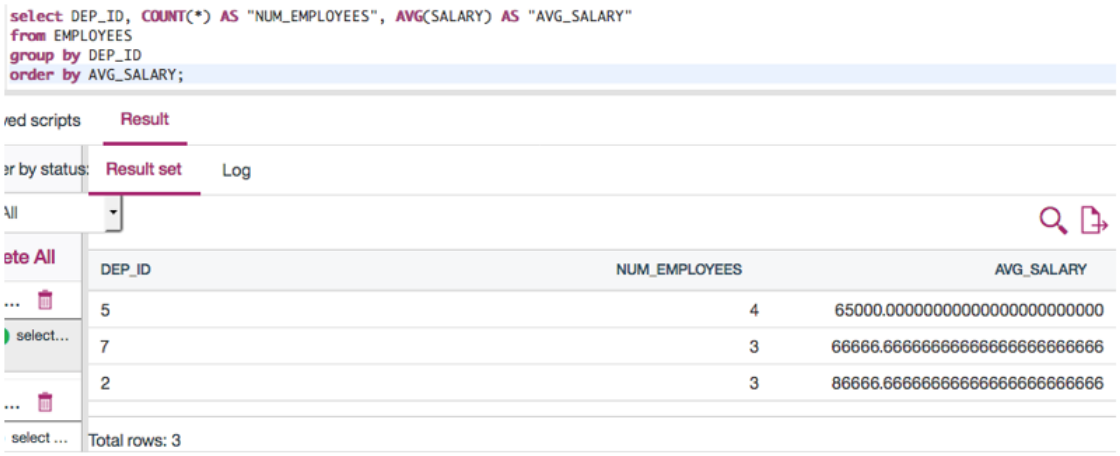
4. Problem:

In SQL problem 3 (Exercise 3 Problem 3), order the result set by Average Salary..

Solution

```
SELECT DEP_ID, COUNT(*) AS "NUM_EMPLOYEES", AVG(SALARY) AS "AVG_SALARY"
FROM EMPLOYEES
GROUP BY DEP_ID
ORDER BY AVG_SALARY;
```

Output



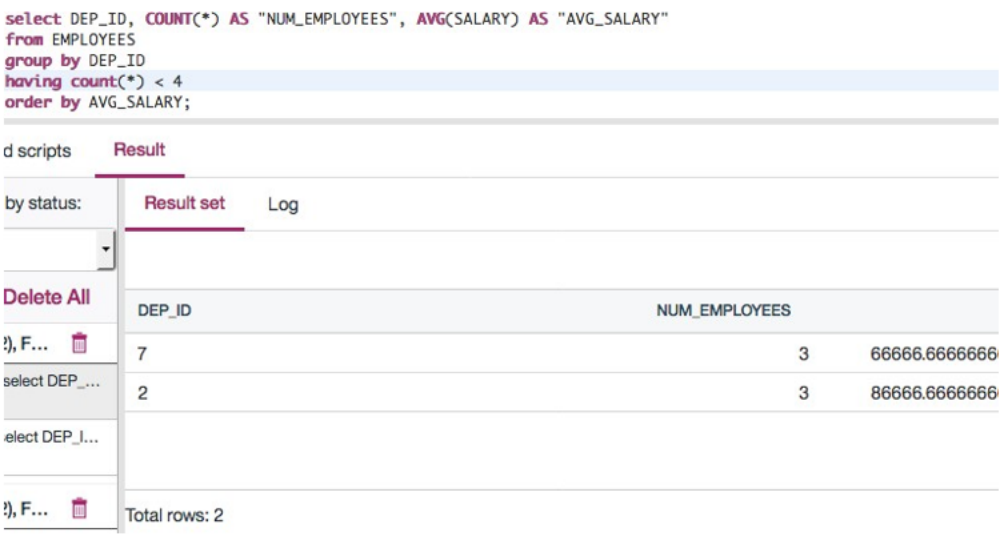
5. Problem:

In SQL problem 4 (Exercise 3 Problem 4), limit the result to departments with fewer than 4 employees.

Solution

```
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;
```

Output



Solution Script

If you would like to run all the solution queries of the SQL problems of this lab with a script, download the script below. Upload the script to the Db2 console and run. Follow Hands-on Lab : Create tables using SQL scripts and Load data into tables on how to upload a script to Db2 console and run it.

- StringPattern-Sorting-Grouping_Solution_Script.sql

Built-in Database Functions

- Most databases come with built-in SQL functions
- Built-in functions can be included as part of SQL statements
- Database functions can significantly reduce the amount of data that needs to be retrieved
- Can speed up data processing

PETRESCUE TABLE

PETRESCUE

ID INTEGER	ANIMAL VARCHAR(20)	QUANTITY INTEGER	COST DECIMAL(6,2)	RESCUEDATE DATE
1	Cat	9	450.09	2018-05-29
2	Dog	3	666.66	2018-06-01
3	Dog	1	100.00	2018-06-04
4	Parrot	2	50.00	2018-06-04
5	Dog	1	75.75	2018-06-10
6	Hamster	6	60.60	2018-06-11
7	Cat	1	44.44	2018-06-11
8	Goldfish	24	48.48	2018-06-14
9	Dog	2	222.22	2018-06-15

Aggregate or Column Functions

- INPUT: Collection of values (e.g. entire column)
- Output: Single value
- Examples: SUM(), MIN(), MAX(), AVG(), etc.

SUM

SUM function: Add up all the values in a column

```
SUM(COLUMN_NAME)
```

Example 1: Add all values in the COST column:

```
select SUM(COST) from PETRESCUE
```

Example 1: Result:

```
1
1718.24
```

When you use an aggregate function, the column in the result set by default is given a number.

Column Alias

Example 2: Explicitly name the output column SUM_OF_COST:

```
select SUM(COST) as SUM_OF_COST
       from PETRESCUE
```

Example 2: Results:

```
SUM_OF_COST
1718.24
```

MIN, MAX

MIN: Return the MINIMUM value
MAX: Return the MAXIMUM value

Example 3A. Get the maximum QUANTITY of any ANIMAL:

```
select MAX(QUANTITY) from PETRESCUE
```

Example 3B. Results:

```
1
24
```

Example 3B. Get the minimum value of ID column for Dogs:

```
select MIN(ID) from PETRESCUE where ANIMAL = 'Dog'
```

Example 3B. Results:

```
1
2
```

Average

AVG() return the average value

Example 4. Specify the Average value of COST:

```
select AVG(COST) from PETRESCUE
```

Example 4. Results:

```
1
190.91555555555555555555555555
```

Average

Mathematical operations can be performed between columns.

Example 5. Calculate the average COST per 'Dog':

```
select AVG(COST / QUANTITY) from PETRESCUE
where ANIMAL = 'Dog'
```

Example 5. Results:

```
1
127.27000000000000000000000000
```

SCALAR and STRING FUNCTIONS

SCALAR: Perform operations on every input value
Examples: ROUND(), LENGTH(), UCASE, LCASE

Example 6: Round UP or DOWN every value in COST:

```
select
    ROUND(COST)
    from PETRESCUE
```

Example 6. Results:

```
1
450.00
667.00
100.00
50.00
76.00
```

UCASE, LCASE

Example 8: Retrieve ANIMAL values in UPPERCASE:

```
select UCASE(ANIMAL) from PETRESCUE
```

Example 8: Results:

```
1
CAT
DOG
DOG
PARROT
DOG
```

UCASE, LCASE

Example 9: Use the function in a WHERE clause :

```
select * from PETRESCUE
where LCASE(ANIMAL) = 'cat'
```

Example 9: Results:

ID	ANIMAL	QUANTITY	COST	DATE
1	Cat	9	450.09	2018-05-29
7	Cat	1	44.44	2018-06-11

UCASE, LCASE

Example 10: Use the DISTINCT() function to get unique values :

```
select DISTINCT(UCASE(ANIMAL)) from PETRESCUE
```

Example 10: Results:

```
1
CAT
DOG
GOLDFISH
HAMSTER
PARROT
```

Date and Time Built-in Functions

Date, Time Functions

Most databases contain special datatypes for dates and times.

DATE: `YYYYMMDD`

TIME: `HHMMSS`

TIMESTAMP: `YYYYXXDDHHMMSSZZZZZZ`

Date / Time functions:

```
YEAR(), MONTH(), DAY(), DAYOFMONTH(), DAYOFWEEK(),
DAYOFYEAR(), WEEK(), HOUR(), MINUTE(), SECOND()
```

Example 11: Extract the DAY portion from a date:

```
select DAY(RESCUEDATE) from PETRESCUE
where ANIMAL='Cat'
```

Example 11: Results:

ID	ANIMAL	QUANTITY	COST	RESCUEDATE
1	Cat	9	450.09	5/29/2018
7	Cat	1	44.44	6/11/2018

```
29
11
```

Example 12: Get the number of rescues during the month of May :

```
select COUNT(*) from PETRESCUE
where MONTH(RESCUEDATE)='05'
```

Example 12: Results:

```
1
```

Date or Time Arithmetic

Example 13: What date is it 3 days after each rescue date?

```
Select (RESCUEDATE + 3 DAYS) from PETRESCUE
```

Example 13: Results:

```
2018-06-01
2018-06-04
2018-06-07
2018-06-07
2018-06-13
```

ID	ANIMAL	QUANTITY	COST	RESCUEDATE	+ 3 DAYS
1	Cat	9	450.09	5/29/2018	6/1/2018
2	Dog	3	666.66	6/1/2018	6/4/2018
3	Dog	1	100	6/4/2018	6/7/2018
4	Parrot	2	50	6/4/2018	6/7/2018
5	Dog	1	75.75	6/10/2018	6/13/2018

Special Registers:

```
CURRENT_DATE, CURRENT_TIME
```

Example 14: Find how many days have passed since each RESCUEDATE till now:

```
Select (CURRENT_DATE - RESCUEDATE) from PETRESCUE
```

Example 14: Sample result (format YMMDD):

```
10921
```

Hands-on Lab: Built-in functions - Aggregate, Scalar, String, Date and Time Functions

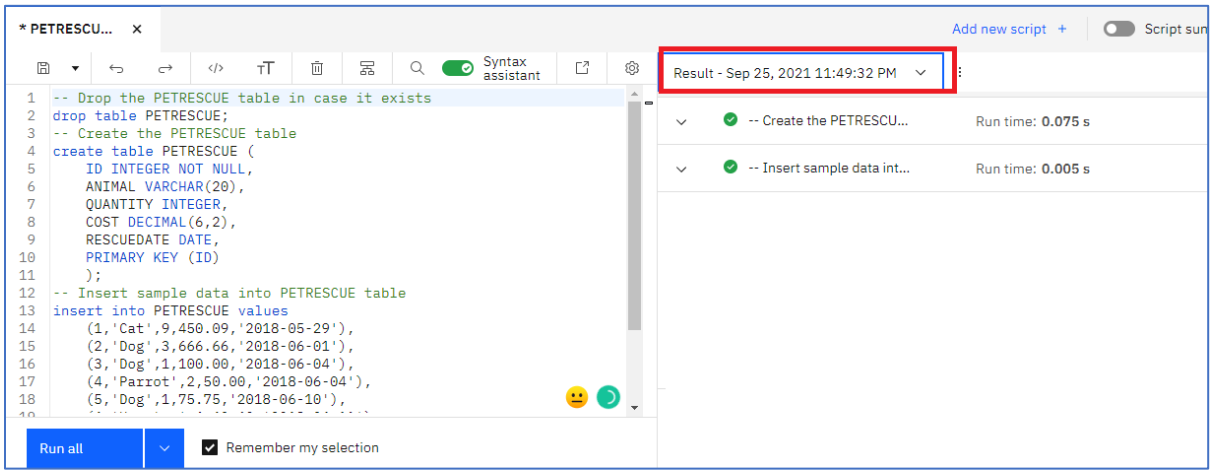
Exercise 1: Create the Pet Rescue table

Rather than create the table manually by typing the DDL commands in the SQL editor, you will execute a script containing the create table command.

1. Download the script file PETRESCUE-CREATE.sql

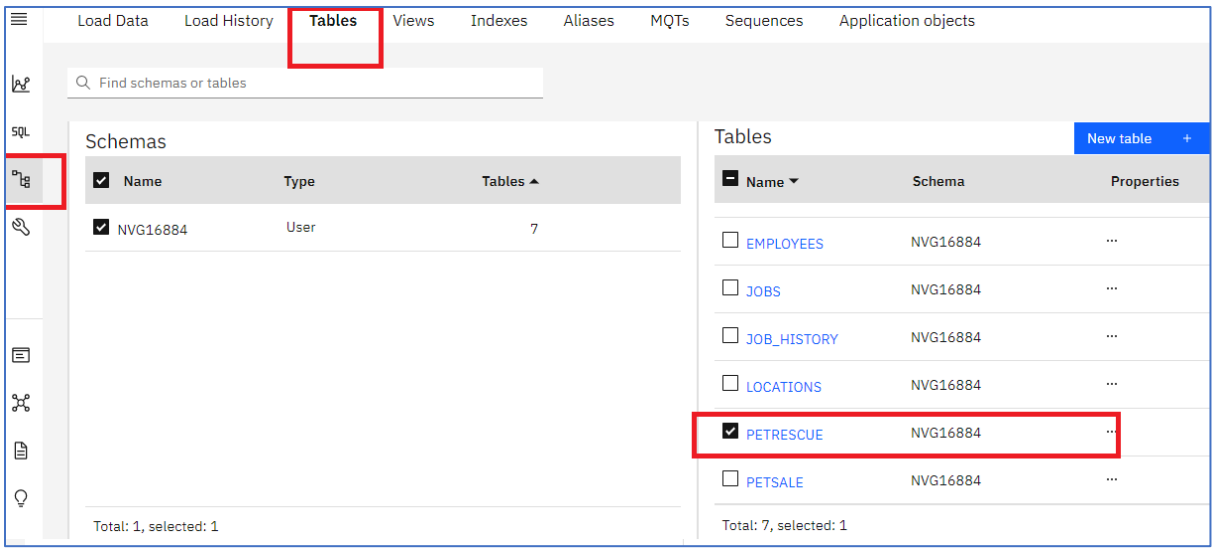
Note: To download, just right-click on the link above and click on **Save As..** or **Save Link As...** depending on your browser. Save the file as a .sql file and not HTML.

2. Login to IBM Cloud and go to the Resources Dashboard: https://cloud.ibm.com/resources where you can find the Db2 service that you created in a previous Lab. Click on the **Db2-xx** service. Next, open the Db2 Console by clicking on **Open Console** button. Go to the **Run SQL** page. The Run SQL tool enables you to run DDL and SQL statements.
3. Click on the + (Add New Script) icon.
4. Click on **From File**.
5. Locate the file **PETRESCUE-CREATE.sql** that you downloaded to your computer earlier and open it.
6. Once the statements are in the SQL Editor tool, you can run the queries against the database by selecting the **Run all** button.
7. On the right side of the SQL editor window, you will see a **Result** section.



Clicking on a query in the Result section will show the execution details of the job - whether it ran successfully or had any errors or warnings. Ensure your queries ran successfully and created all the tables.

8. Now you can look at the tables you created. Navigate to the three-bar menu icon, select **Explore**, then click on **Tables**.



9. Select the Schema corresponding to your Db2 userid. Then on the right side of the screen you should see the newly created **PETRESCUE** table listed (plus any other tables you may have created in previous labs e.g. INSTRUCTOR, TEST, etc.).
10. Click on any of the tables and you will see its SCHEMA definition (that is list of columns, their data types, and so on). You can also click **View Data** to view the content of the table.

NVG16884.PETRESCUE

ID	ANIMAL	QUANTITY	COST	RESCUEDATE
1	Cat	9	450.09	2018-05-29
2	Dog	3	666.66	2018-06-01
3	Dog	1	100.00	2018-06-04
4	Parrot	2	50.00	2018-06-04
5	Dog	1	75.75	2018-06-10
6	Hamster	6	60.60	2018-06-11
7	Cat	1	44.44	2018-06-11
8	Goldfish	24	48.48	2018-06-14
9	Dog	2	222.22	2018-06-15

Exercise 2: Aggregate Functions

Query A1: Enter a function that calculates the total cost of all animal rescues in the PETRESCUE table.

```
select SUM(COST) from PETRESCUE;
```

Query A2: Enter a function that displays the total cost of all animal rescues in the PETRESCUE table in a column called SUM_OF_COST.

```
select SUM(COST) AS SUM_OF_COST from PETRESCUE;
```

Query A3: Enter a function that displays the maximum quantity of animals rescued.

```
select MAX(QUANTITY) from PETRESCUE;
```

Query A4: Enter a function that displays the average cost of animals rescued.

```
select AVG(COST) from PETRESCUE;
```

Query A5: Enter a function that displays the average cost of rescuing a dog. *Hint - Bear in my the cost of rescuing one dog on day, is different from another day. So you will have to use and average of averages.*

```
select AVG(COST/QUANTITY) from PETRESCUE where ANIMAL = 'Dog';
```

Exercise 3: Scalar and String Functions

Query B1: Enter a function that displays the rounded cost of each rescue.

```
select ROUND(COST) from PETRESCUE;
```

Query B2: Enter a function that displays the length of each animal name.

```
select LENGTH(ANIMAL) from PETRESCUE;
```

Query B3: Enter a function that displays the animal name in each rescue in uppercase.

```
select UCASE(ANIMAL) from PETRESCUE;
```

Query B4: Enter a function that displays the animal name in each rescue in uppercase without duplications.

```
select DISTINCT(UCASE(ANIMAL)) from PETRESCUE;
```

Query B5: Enter a query that displays all the columns from the PETRESCUE table, where the animal(s) rescued are cats. Use **cat** in lower case in the query.

```
select * from PETRESCUE where LCASE(ANIMAL) = 'cat';
```

Exercise 4: Date and Time Functions

Query C1: Enter a function that displays the day of the month when cats have been rescued.

```
select DAY(RESCUEDATE) from PETRESCUE where ANIMAL = 'Cat';
```

Query C2: Enter a function that displays the number of rescues on the 5th month.

```
select SUM(QUANTITY) from PETRESCUE where MONTH(RESCUEDATE)='05';
```

Query C3: Enter a function that displays the number of rescues on the 14th day of the month.

```
select SUM(QUANTITY) from PETRESCUE where DAY(RESCUEDATE)='14';
```

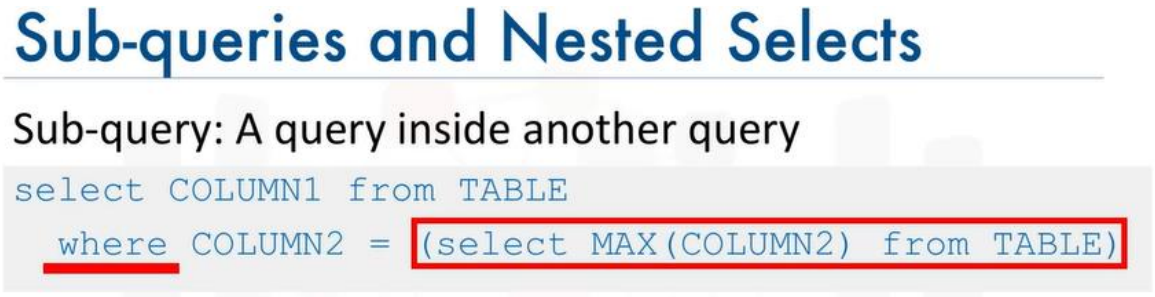
Query C4: Animals rescued should see the vet within three days of arrivals. Enter a function that displays the third day from each rescue.

```
select (RESCUEDATE + 3 DAYS) from PETRESCUE;
```

Query C5: Enter a function that displays the length of time the animals have been rescued; the difference between today's date and the recue date.

```
select (CURRENT DATE - RESCUEDATE) from PETRESCUE;
```

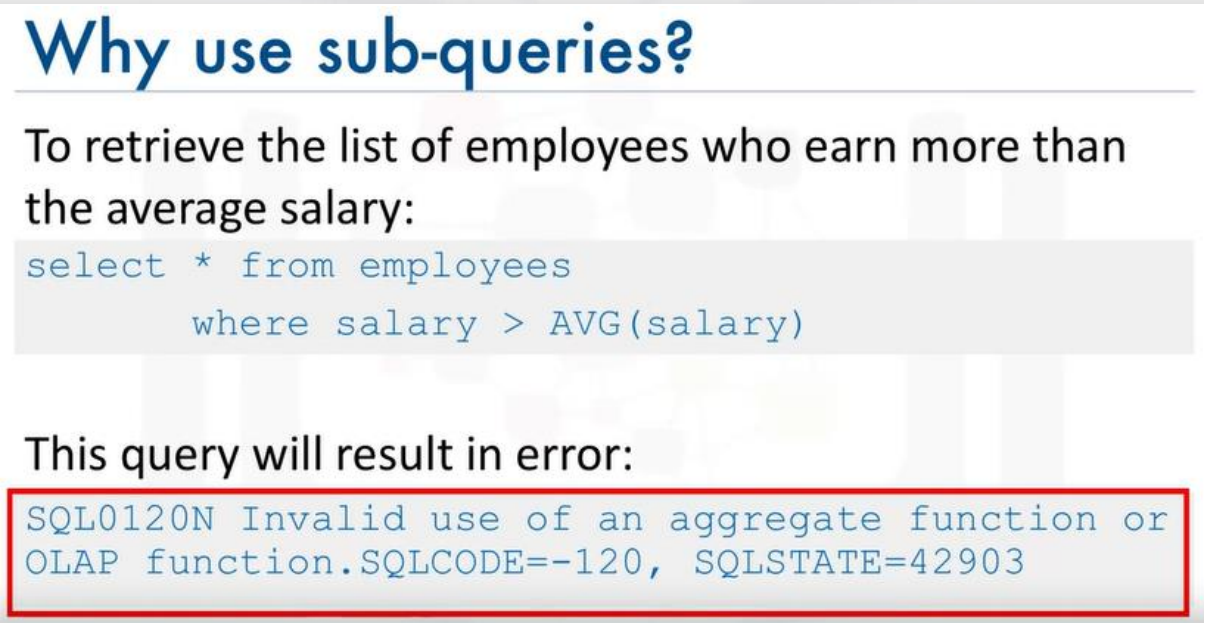
Sub-Queries and Nested Selects



Consider the employees table from the previous video. The first few rows of data are shown here. The table contains several columns, including an employee ID, first name, last name, salary, etc. We will now go over some examples involving this table.

EMPLOYEES

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1001	John	Thomas	123456	1976-01-09	M	5631 Rice, OakPark,IL	100	100000	30001	2
E1002	Alice	James	123457	1972-07-31	F	980 Berry ln, Elgin,IL	200	80000	30002	5
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs, Gary,IL	300	50000	30002	5



Running this query will result in an error like the one shown. Indicating an invalid use of the aggregate function. One of the limitations of built-in aggregate functions, like the average function, is that they cannot always be evaluated in the WHERE clause.

So to evaluate a function like average in the WHERE clause, we can make use of a sub-select expression like the one shown here.

Sub-queries to evaluate Aggregate functions

- Cannot evaluate Aggregate functions like AVG() in the WHERE clause –
- Therefore, use a sub-Select expression:

```
select EMP_ID, F_NAME, L_NAME, SALARY
       from employees
       where SALARY <
       (select AVG(SALARY) from employees);
```

Result:

EMP_ID	F_NAME	L_NAME	SALARY
E1003	Steve	Wells	50000.00
E1004	Santosh	Kumar	60000.00
E1007	Mary	Thomas	65000.00

Sub-queries in list of columns

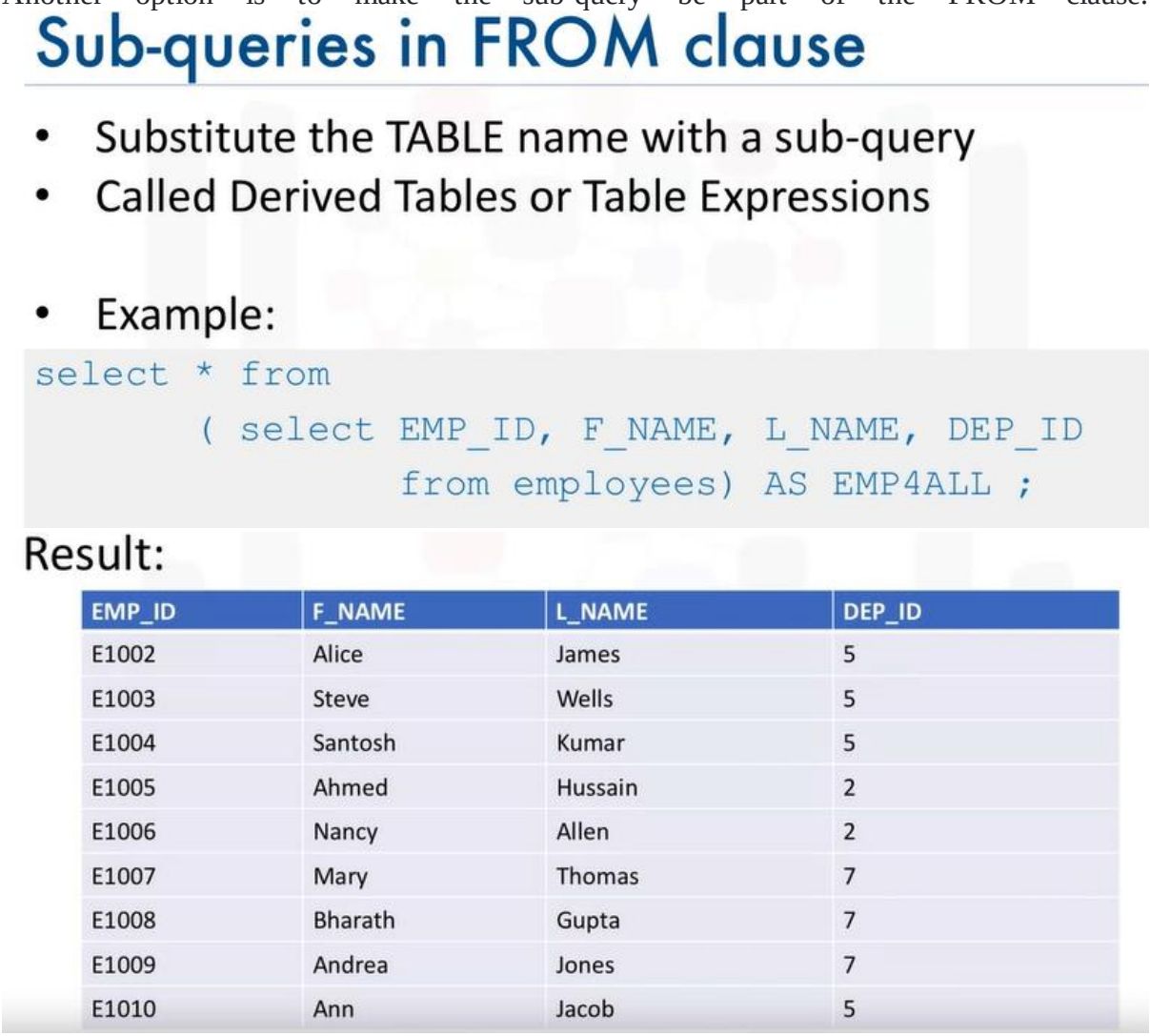
- Substitute column name with a sub-query
- Called Column Expressions

```
select EMP_ID, SALARY, AVG(SALARY) AS AVG_SALARY
       from employees ;
```

Running this query will result in an error indicating that no group by clause is specified.

```
select EMP_ID, SALARY,
       ( select AVG(SALARY) from employees )
                AS AVG_SALARY
       from employees ;
```

Another option is to make the sub-query be part of the FROM clause.



EMP_ID	F_NAME	L_NAME	DEP_ID
E1002	Alice	James	5
E1003	Steve	Wells	5
E1004	Santosh	Kumar	5
E1005	Ahmed	Hussain	2
E1006	Nancy	Allen	2
E1007	Mary	Thomas	7
E1008	Bharath	Gupta	7
E1009	Andrea	Jones	7
E1010	Ann	Jacob	5

Hands-on Lab: Sub-queries and Nested SELECTs
How does a typical Nested SELECT statement syntax look?

```
SELECT column_name [,column_name ]
FROM table1 [, table2 ]
WHERE column_name OPERATOR
  (SELECT column_name [, column_name ]
  FROM table1 [, table2 ]
  WHERE condition);
```

NOTE: This lab requires you to have all 5 of these tables of the HR database populated with sample data on Db2. If you didn't complete the earlier lab in this module, you won't have the

tables above populated with sample data on Db2, so you will need to go through the lab below first:

- Hands-on Lab : Create tables using SQL scripts and Load data into tables

Exercise:

1. Problem:

   Execute a failing query (i.e. one which gives an error) to retrieve all employees records whose salary is lower than the average salary.

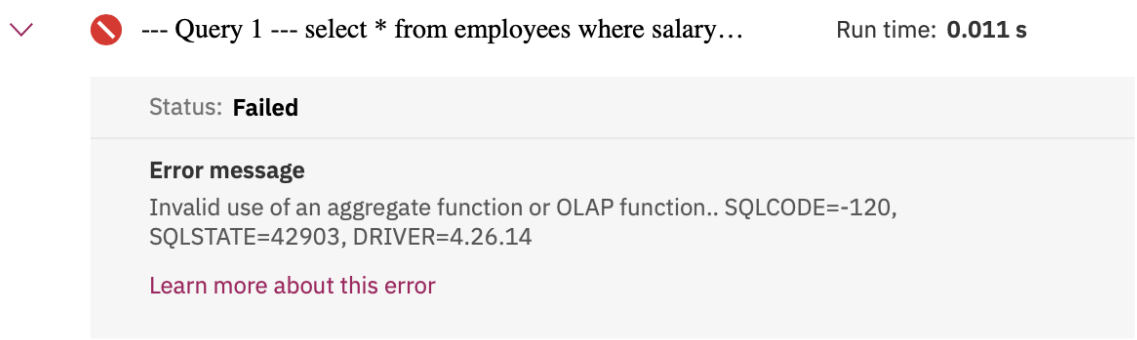
   Hint

   Use the AVG aggregate function.

   Solution

```
select *
from employees
where salary < AVG(salary);
```

   Output



2. Problem:

   Execute a working query using a sub-select to retrieve all employees records whose salary is lower than the average salary.

   Hint

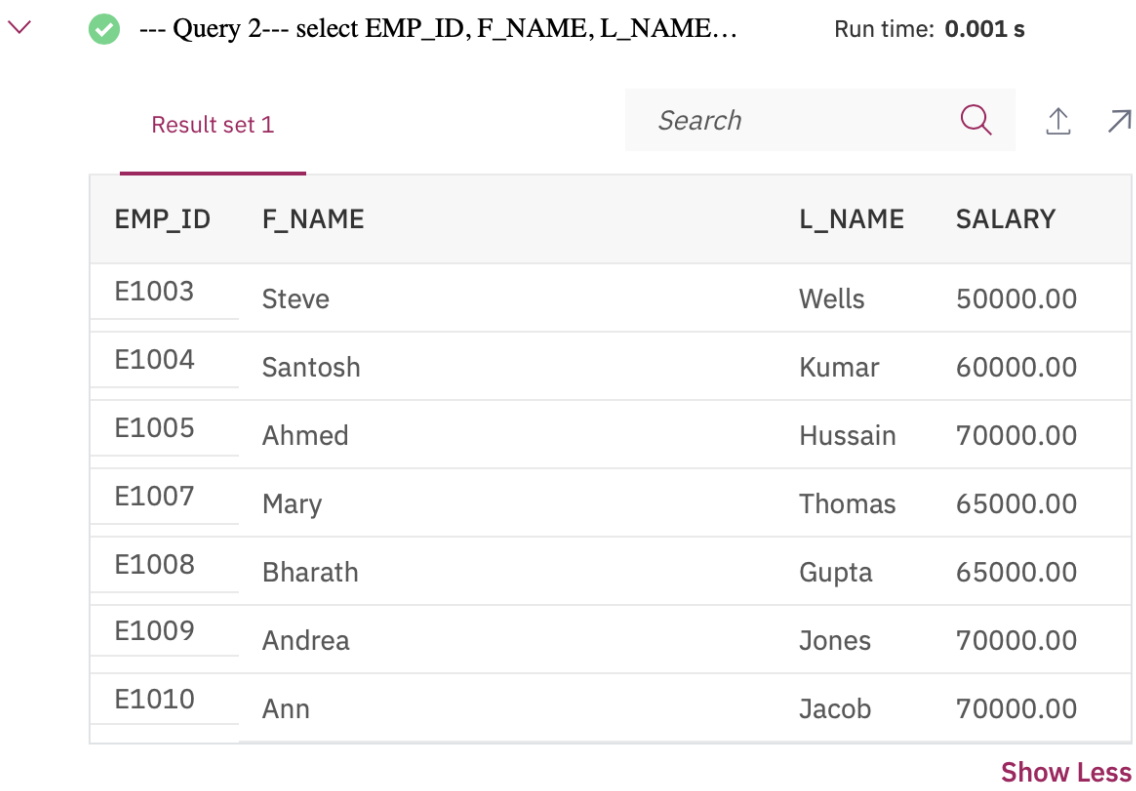
   Put AVG(SALARY) of the inner SELECT in comparison with SALARY of the outer SELECT.

   Solution

```
select EMP_ID, F_NAME, L_NAME, SALARY
```

```
from employees
where SALARY  < (select AVG(SALARY)
            from employees);
```

Output



EMP_ID	F_NAME	L_NAME	SALARY
E1003	Steve	Wells	50000.00
E1004	Santosh	Kumar	60000.00
E1005	Ahmed	Hussain	70000.00
E1007	Mary	Thomas	65000.00
E1008	Bharath	Gupta	65000.00
E1009	Andrea	Jones	70000.00
E1010	Ann	Jacob	70000.00

3. Problem:

Execute a failing query (i.e. one which gives an error) to retrieve all employees records with EMP_ID, SALARY and maximum salary as MAX_SALARY in every row.

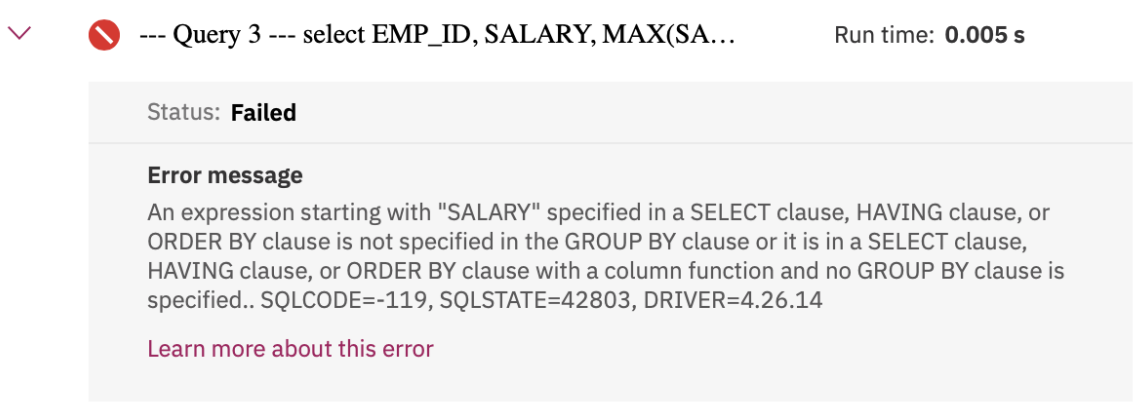
Hint

Use the MAX aggregate function.

Solution

```
select EMP_ID, SALARY,  MAX(SALARY)  AS MAX_SALARY
from employees;
```

Output



4. Problem:

   Execute a Column Expression that retrieves all employees records with EMP_ID, SALARY and maximum salary as MAX_SALARY in every row.

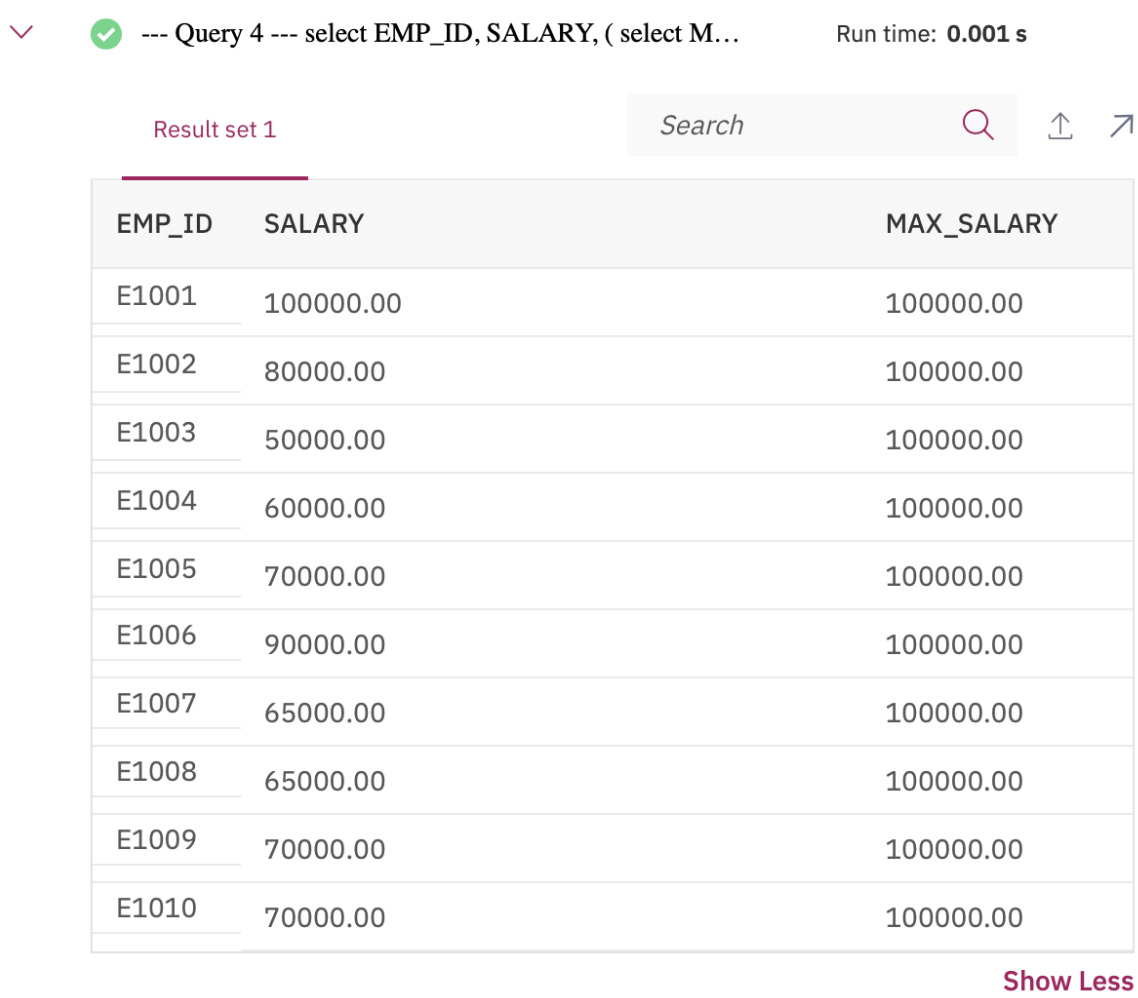
   Hint

   Use the SELECT(which retrieves MAX(SALARY)) as a column of the other SELECT.

   Solution

```
select EMP_ID, SALARY, ( select MAX(SALARY) from employees ) AS MAX_SALARY
from employees;
```

   Output



EMP_ID	SALARY	MAX_SALARY
E1001	100000.00	100000.00
E1002	80000.00	100000.00
E1003	50000.00	100000.00
E1004	60000.00	100000.00
E1005	70000.00	100000.00
E1006	90000.00	100000.00
E1007	65000.00	100000.00
E1008	65000.00	100000.00
E1009	70000.00	100000.00
E1010	70000.00	100000.00

Show Less

5. Problem:

   Execute a Table Expression for the EMPLOYEES table that excludes columns with sensitive employee data (i.e. does not include columns: SSN, B_DATE, SEX, ADDRESS, SALARY).

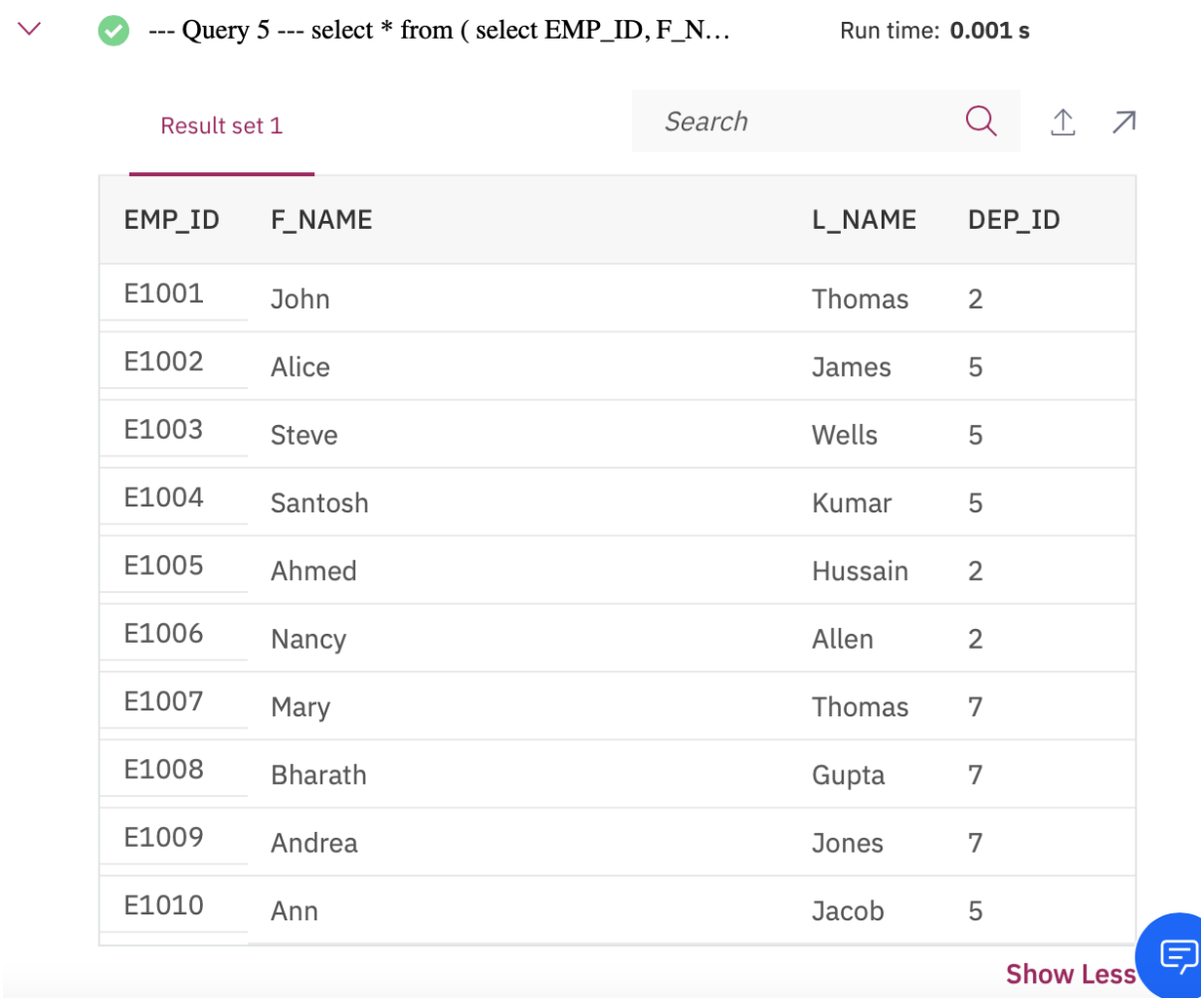
   Hint

   Use a SELECT (which retrieves non-sensitive employee data) after FROM of the other SELECT.

   Solution

```
select * from ( select EMP_ID, F_NAME, L_NAME, DEP_ID from employees) AS EMP4ALL;
```

   Output



EMP_ID	F_NAME	L_NAME	DEP_ID
E1001	John	Thomas	2
E1002	Alice	James	5
E1003	Steve	Wells	5
E1004	Santosh	Kumar	5
E1005	Ahmed	Hussain	2
E1006	Nancy	Allen	2
E1007	Mary	Thomas	7
E1008	Bharath	Gupta	7
E1009	Andrea	Jones	7
E1010	Ann	Jacob	5

Solution Script

If you would like to run all the solution queries of the SQL problems in this lab with a script, download the script below. Upload the script to the Db2 console and run it. Follow Hands-on Lab : Create tables using SQL scripts and Load data into tables on how to upload a script to Db2 console and run it.

- SubQueries_Solution_Script.sql

Working with Multiple Tables

Working with Multiple Tables

Ways to access multiple tables in the same query:

1. Sub-queries
2. Implicit JOIN
3. JOIN operators (INNER JOIN, OUTER JOIN, etc.)

Tables for Examples in this Lesson

EMPLOYEES:

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1001	John	Thomas	123456	1976-01-09	M	5631 Rice, OakPark,IL	100	100000	30001	2
E1002	Alice	James	123457	1972-07-31	F	980 Berry ln, Elgin,IL	200	80000	30002	5
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs, Gary,IL	300	50000	30002	5

DEPARTMENTS:

DEPT_ID_DEP	DEP_NAME	MANAGER_ID	LOC_ID
5	Software Development	30002	L0002
7	Design Team	30003	L0003

Accessing Multiple Tables with Sub-queries

To retrieve only the employee records that correspond to departments in the DEPARTMENTS table:

```
select * from employees
       where DEP_ID IN
       ( select DEPT_ID_DEP from departments );
```

Now, let's use sub-queries to work with multiple tables. If we want to retrieve only the employee records from the employees table for which a department ID exists in the departments table, we can use a sub-query as follows. Select star from employees, where department_ID IN, select department_ID_department from departments. Here the outer query accesses the employees table and the sub-query on the departments table is used for filtering the result set of the outer query.

Result:

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1002	Alice	James	123457	7/31/1972	F	980 Berry ln, Elgin,IL	200	80000	30002	5
E1003	Steve	Wells	123458	8/10/1980	M	291 Springs, Gary,IL	300	50000	30002	5
E1004	Santosh	Kumar	123459	7/20/1985	M	511 Aurora Av, Aurora,IL	400	60000	30004	5
E1007	Mary	Thomas	123412	5/5/1975	F	100 Rose Pl, Gary,IL	650	65000	30003	7
E1008	Bharath	Gupta	123413	5/6/1985	M	145 Berry Ln, Naperville,IL	660	65000	30003	7
E1009	Andrea	Jones	123414	7/9/1990	F	120 Fall Creek, Gary,IL	234	70000	30003	7
E1010	Ann	Jacob	123415	3/30/1982	F	111 Britany Springs,Elgin,IL	220	70000	30004	5

Multiple Tables with Sub-queries

To retrieve only the list of employees from a specific location:

- EMPLOYEES table does not contain location information
- Need to get location info from DEPARTMENTS table

```
select * from employees
        where DEP_ID IN
        ( select DEPT_ID_DEP from departments
                where LOC_ID = 'L0002' );
```

Result:

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1002	Alice	James	123457	7/31/1972	F	980 Berry ln, Elgin,IL	200	80000	30002	5
E1003	Steve	Wells	123458	8/10/1980	M	291 Springs, Gary,IL	300	50000	30002	5
E1004	Santosh	Kumar	123459	7/20/1985	M	511 Aurora Av, Aurora,IL	400	60000	30004	5
E1010	Ann	Jacob	123415	3/30/1982	F	111 Britany Springs,Elgin,IL	220	70000	30004	5

To 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 ) ;
```

Result:

DEPT_ID_DEP	DEP_NAME
5	Software Group

We can also access multiple tables by specifying them in the FROM clause of the query.

Accessing multiple tables with Implicit Join

Specify 2 tables in the FROM clause:

```
select * from employees, departments;
```

The result is a full join (or Cartesian join):

- Every row in the first table is joined with every row in the second table
- The result set will have more rows than in both tables

Result:

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID	DEPT_ID_DEP	DEP_NAME	MANAGER_ID	LOC_ID
E1002	Alice	James	123457	7/31/1972	F	980 Berry ln, Elgin,IL	200	80000	30002	5	5	Software Group	30002	L0002
E1003	Steve	Wells	123458	8/10/1980	M	291 Springs, Gary,IL	300	50000	30002	5	5	Software Group	30002	L0002
E1004	Santosh	Kumar	123459	7/20/1985	M	511 Aurora Av, Aurora,IL	400	60000	30002	5	5	Software Group	30002	L0002
E1005	Ahmed	Hussain	123410	1/4/1981	M	216 Oak Tree, Geneva,IL	500	70000	30002	2	5	Software Group	30002	L0002
E1006	Nancy	Allen	123411	2/6/1978	F	111 Green Pl, Elgin,IL	600	90000	30002	2	5	Software Group	30002	L0002
E1005	Ahmed			1/4/1981	M		500	70000				Team		
E1006	Nancy	Allen	123411	2/6/1978	F	111 Green Pl, Elgin,IL	600	90000	30003	2	7	Design Team	30003	L0003
E1007	Mary	Thomas	123412	5/5/1975	F	100 Rose Pl, Gary,IL	650	65000	30003	7	7	Design Team	30003	L0003
E1008	Bharath	Gupta	123413	5/6/1985	M	145 Berry Ln, Naperville,IL	660	65000	30003	7	7	Design Team	30003	L0003
E1009	Andrea	Jones	123414	7/9/1990	F	120 Fall Creek, Gary,IL	234	70000	30003	7	7	Design Team	30003	L0003
E1010	Ann	Jacob	123415	3/30/1982	F	111 Britany Springs,Elgin,IL	220	70000	30003	5	7	Design Team	30003	L0003

Use additional operands to limit the result set:

```
select * from employees, departments
        where employees.DEP_ID =
              departments.DEPT_ID_DEP;
```

Notice that in the WHERE clause, we prefix the name of the column with the name of the table. This is to fully qualify the column name, since it's possible that different tables could have some column names that are exactly the same. Since the table names can sometimes be long, we can use shorter aliases for table names as shown here.

Use shorter aliases for table names:

```
select * from employees E, departments D
        where E.DEP_ID = D.DEPT_ID_DEP;
```

Result:

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID	DEPT_ID_DEP	DEP_NAME	MANAGER_ID	LOC_ID
E1002	Alice	James	123457	7/31/1972	F	980 Berry ln, Elgin,IL	200	80000	30002	5	5	Software Group	30002	L0002
E1003	Steve	Wells	123458	8/10/1980	M	291 Springs, Gary,IL	300	50000	30002	5	5	Software Group	30002	L0002
E1004	Santosh	Kumar	123459	7/20/1985	M	511 Aurora Av, Aurora,IL	400	60000	30002	5	5	Software Group	30002	L0002
E1007	Mary	Thomas	123412	5/5/1975	F	100 Rose Pl, Gary,IL	650	65000	30003	7	7	Design Team	30003	L0003
E1008	Bharath	Gupta	123413	5/6/1985	M	145 Berry Ln, Naperville,IL	660	65000	30003	7	7	Design Team	30003	L0003
E1009	Andrea	Jones	123414	7/9/1990	F	120 Fall Creek, Gary,IL	234	70000	30003	7	7	Design Team	30003	L0003
E1010	Ann	Jacob	123415	3/30/1982	F	111 Britany Springs,Elgin,IL	220	70000	30002	5	5	Software Group	30002	L0002

To see the department name for each employee:

```
select EMP_ID, DEP_NAME
        from employees E, departments D
        where E.DEP_ID = D.DEPT_ID_DEP;
```

Result:

EMP_ID	DEP_NAME
E1002	Software Group
E1003	Software Group
E1004	Software Group
E1007	Design Team
E1008	Design Team
E1009	Design Team
E1010	Software Group

Column names in the select clause can be pre-fixed by aliases:

```
select E.EMP_ID, D.DEP_ID_DEP from
       employees E, departments D
       where E.DEP_ID = D.DEPT_ID_DEP
```

Result:

EMP_ID	DEPT_ID_DEP
E1002	5
E1003	5
E1004	5
E1005	2
E1006	2
E1007	7
E1008	7
E1009	7
E1010	5

Hands-on Lab: Working with Multiple Tables

How does an Implicit version of CROSS JOIN (also known as Cartesian Join) statement syntax look?

```
SELECT column_name(s)
FROM table1, table2;
```

How does an Implicit version of INNER JOIN statement syntax look?

```
SELECT column_name(s)
FROM table1, table2
WHERE table1.column_name = table2.column_name;
```

Exercise 1: Accessing Multiple Tables with Sub-Queries

1. Problem:

   Retrieve only the EMPLOYEES records that correspond to jobs in the JOBS table.

   Solution

   ```
   select * from employees where JOB_ID IN (select JOB_IDENT from jobs);
   ```

   Output

Result set 1 Search

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1001	John	Thomas	123456	1976-01-09	M	5631 Rice, OakPark,IL	100	100000.00	30001	2
E1002	Alice	James	123457	1972-07-31	F	980 Berry ln, Elgin,IL	200	80000.00	30002	5
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs, Gary,IL	300	50000.00	30002	5
E1004	Santosh	Kumar	123459	1985-07-20	M	511 Aurora Av, Aurora,IL	400	60000.00	30004	5
E1005	Ahmed	Hussain	123410	1981-01-04	M	216 Oak Tree, Geneva,IL	500	70000.00	30001	2
E1006	Nancy	Allen	123411	1978-02-06	F	111 Green Pl, Elgin,IL	600	90000.00	30001	2
E1007	Mary	Thomas	123412	1975-05-05	F	100 Rose Pl, Gary,IL	650	65000.00	30003	7
E1008	Bharath	Gupta	123413	1985-05-06	M	145 Berry Ln, Naperville,IL	660	65000.00	30003	7
E1009	Andrea	Jones	123414	1990-07-09	F	120 Fall Creek, Gary,IL	234	70000.00	30003	7
E1010	Ann	Jacob	123415	1982-03-30	F	111 Britany Springs,Elgin,IL	220	70000.00	30004	5

2. Problem:

   Retrieve only the list of employees whose JOB_TITLE is Jr. Designer.

   Solution

```
select * from employees where JOB_ID IN (select JOB_IDENT from jobs where JOB_TITLE= 'Jr. Designer');
```

   Output

Result set 1 Search

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID
E1007	Mary	Thomas	123412	1975-05-05	F	100 Rose Pl, Gary,IL	650	65000.00	30003	7
E1008	Bharath	Gupta	123413	1985-05-06	M	145 Berry Ln, Naperville,IL	660	65000.00	30003	7

3. Problem:

   Retrieve JOB information and list of employees who earn more than $70,000.

   Solution

```
select JOB_TITLE, MIN_SALARY,MAX_SALARY,JOB_IDENT  from jobs where JOB_IDENT IN (select JOB_ID from employees where SALARY > 70000 );
```

   Output

Result set 1 Search

JOB_TITLE	MIN_SALARY	MAX_SALARY	JOB_IDENT
Sr. Architect	60000.00	100000.00	100
Sr.Software Dev	60000.00	80000.00	200
Lead Architect	70000.00	100000.00	600

4. Problem:

   Retrieve JOB information and list of employees whose birth year is after 1976.

   Solution

```
select JOB_TITLE, MIN_SALARY,MAX_SALARY,JOB_IDENT  from jobs where JOB_IDENT IN (select JOB_ID from employees where YEAR(B_DATE)>1976 );
```

Output

JOB_TITLE	MIN_SALARY	MAX_SALARY	JOB_IDENT
Sr. Designer	70000.00	90000.00	220
Sr. Designer	70000.00	90000.00	234
Jr.Software Dev	40000.00	60000.00	300
Jr.Software Dev	40000.00	60000.00	400
Jr. Architect	50000.00	70000.00	500
Lead Architect	70000.00	100000.00	600
Jr. Designer	60000.00	70000.00	660

5. Problem:

Retrieve JOB information and list of female employees whose birth year is after 1976.

Solution

```
select JOB_TITLE, MIN_SALARY,MAX_SALARY,JOB_IDENT  from jobs where JOB_IDENT IN
(select JOB_ID from employees where YEAR(B_DATE)>1976  and SEX='F' );
```

Output

JOB_TITLE	MIN_SALARY	MAX_SALARY	JOB_IDENT
Sr. Designer	70000.00	90000.00	220
Sr. Designer	70000.00	90000.00	234
Lead Architect	70000.00	100000.00	600

Exercise 2: Accessing Multiple Tables with Implicit Joins

1. Problem:

Perform an implicit cartesian/cross join between EMPLOYEES and JOBS tables.

Solution

```
select * from employees, jobs;
```

Output

Result set 1

Search

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID	JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY
E1001	John	Thomas	123456	1976-01-09	M	5631 Rice, OakPark,IL	100	100000.00	30001	2	100	Sr. Architect	60000.00	100000.00
E1002	Alice	James	123457	1972-07-31	F	980 Berry ln, Elgin,IL	200	80000.00	30002	5	100	Sr. Architect	60000.00	100000.00
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs, Gary,IL	300	50000.00	30002	5	100	Sr. Architect	60000.00	100000.00
E1004	Santosh	Kumar	123459	1985-07-20	M	511 Aurora Av, Aurora,IL	400	60000.00	30004	5	100	Sr. Architect	60000.00	100000.00
E1005	Ahmed	Hussain	123410	1981-01-04	M	216 Oak Tree, Geneva,IL	500	70000.00	30001	2	100	Sr. Architect	60000.00	100000.00
E1006	Nancy	Allen	123411	1978-02-06	F	111 Green Pl, Elgin,IL	600	90000.00	30001	2	100	Sr. Architect	60000.00	100000.00
E1007	Mary	Thomas	123412	1975-05-05	F	100 Rose Pl, Gary,IL	650	65000.00	30003	7	100	Sr. Architect	60000.00	100000.00
E1008	Bharath	Gupta	123413	1985-05-06	M	145 Berry Ln, Naperville,IL	660	65000.00	30003	7	100	Sr. Architect	60000.00	100000.00
E1009	Andrea	Jones	123414	1990-07-09	F	120 Fall Creek, Gary,IL	234	70000.00	30003	7	100	Sr. Architect	60000.00	100000.00
E1010	Ann	Jacob	123415	1982-03-30	F	111 Britany Springs,Elgin,IL	220	70000.00	30004	5	100	Sr. Architect	60000.00	100000.00
E1001	John	Thomas	123456	1976-01-09	M	5631 Rice, OakPark,IL	100	100000.00	30001	2	200	Sr.Software Dev	60000.00	80000.00
E1002	Alice	James	123457	1972-07-31	F	980 Berry ln, Elgin,IL	200	80000.00	30002	5	200	Sr.Software Dev	60000.00	80000.00
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs, Gary,IL	300	50000.00	30002	5	200	Sr.Software Dev	60000.00	80000.00
E1004	Santosh	Kumar	123459	1985-07-20	M	511 Aurora Av, Aurora,IL	400	60000.00	30004	5	200	Sr.Software Dev	60000.00	80000.00
E1005	Ahmed	Hussain	123410	1981-01-04	M	216 Oak Tree, Geneva,IL	500	70000.00	30001	2	200	Sr.Software Dev	60000.00	80000.00
E1006	Nancy	Allen	123411	1978-02-06	F	111 Green Pl, Elgin,IL	600	90000.00	30001	2	200	Sr.Software Dev	60000.00	80000.00
E1007	Mary	Thomas	123412	1975-05-05	F	100 Rose Pl, Gary,IL	650	65000.00	30003	7	200	Sr.Software Dev	60000.00	80000.00
E1008	Bharath	Gupta	123413	1985-05-06	M	145 Berry Ln, Naperville,IL	660	65000.00	30003	7	200	Sr.Software Dev	60000.00	80000.00
E1009	Andrea	Jones	123414	1990-07-09	F	120 Fall Creek, Gary,IL	234	70000.00	30003	7	200	Sr.Software Dev	60000.00	80000.00
E1010	Ann	Jacob	123415	1982-03-30	F	111 Britany Springs,Elgin,IL	220	70000.00	30004	5	200	Sr.Software Dev	60000.00	80000.00

2. Problem:

Retrieve only the EMPLOYEES records that correspond to jobs in the JOBS table.

Solution

```
select * from employees, jobs where employees.JOB_ID = jobs.JOB_IDENT;
```

Output

Result set 1

Search

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID	JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY
E1001	John	Thomas	123456	1976-01-09	M	5631 Rice, OakPark,IL	100	100000.00	30001	2	100	Sr. Architect	60000.00	100000.00
E1002	Alice	James	123457	1972-07-31	F	980 Berry ln, Elgin,IL	200	80000.00	30002	5	200	Sr.Software Dev	60000.00	80000.00
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs, Gary,IL	300	50000.00	30002	5	300	Jr.Software Dev	40000.00	60000.00
E1004	Santosh	Kumar	123459	1985-07-20	M	511 Aurora Av, Aurora,IL	400	60000.00	30004	5	400	Jr.Software Dev	40000.00	60000.00
E1005	Ahmed	Hussain	123410	1981-01-04	M	216 Oak Tree, Geneva,IL	500	70000.00	30001	2	500	Jr. Architect	50000.00	70000.00
E1006	Nancy	Allen	123411	1978-02-06	F	111 Green Pl, Elgin,IL	600	90000.00	30001	2	600	Lead Architect	70000.00	100000.00
E1007	Mary	Thomas	123412	1975-05-05	F	100 Rose Pl, Gary,IL	650	65000.00	30003	7	650	Jr. Designer	60000.00	70000.00
E1008	Bharath	Gupta	123413	1985-05-06	M	145 Berry Ln, Naperville,IL	660	65000.00	30003	7	660	Jr. Designer	60000.00	70000.00
E1009	Andrea	Jones	123414	1990-07-09	F	120 Fall Creek, Gary,IL	234	70000.00	30003	7	234	Sr. Designer	70000.00	90000.00
E1010	Ann	Jacob	123415	1982-03-30	F	111 Britany Springs,Elgin,IL	220	70000.00	30004	5	220	Sr. Designer	70000.00	90000.00

3. Problem:

Redo the previous query, using shorter aliases for table names.

Solution

```
select * from employees E, jobs J where E.JOB_ID = J.JOB_IDENT;
```

Output

Result set 1

Search

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS	JOB_ID	SALARY	MANAGER_ID	DEP_ID	JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY
E1001	John	Thomas	123456	1976-01-09	M	5631 Rice, OakPark,IL	100	100000.00	30001	2	100	Sr. Architect	60000.00	100000.00
E1002	Alice	James	123457	1972-07-31	F	980 Berry ln, Elgin,IL	200	80000.00	30002	5	200	Sr.Software Dev	60000.00	80000.00
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs, Gary,IL	300	50000.00	30002	5	300	Jr.Software Dev	40000.00	60000.00
E1004	Santosh	Kumar	123459	1985-07-20	M	511 Aurora Av, Aurora,IL	400	60000.00	30004	5	400	Jr.Software Dev	40000.00	60000.00
E1005	Ahmed	Hussain	123410	1981-01-04	M	216 Oak Tree, Geneva,IL	500	70000.00	30001	2	500	Jr. Architect	50000.00	70000.00
E1006	Nancy	Allen	123411	1978-02-06	F	111 Green Pl, Elgin,IL	600	90000.00	30001	2	600	Lead Architect	70000.00	100000.00
E1007	Mary	Thomas	123412	1975-05-05	F	100 Rose Pl, Gary,IL	650	65000.00	30003	7	650	Jr. Designer	60000.00	70000.00
E1008	Bharath	Gupta	123413	1985-05-06	M	145 Berry Ln, Naperville,IL	660	65000.00	30003	7	660	Jr. Designer	60000.00	70000.00
E1009	Andrea	Jones	123414	1990-07-09	F	120 Fall Creek, Gary,IL	234	70000.00	30003	7	234	Sr. Designer	70000.00	90000.00
E1010	Ann	Jacob	123415	1982-03-30	F	111 Britany Springs,Elgin,IL	220	70000.00	30004	5	220	Sr. Designer	70000.00	90000.00

4. Problem:

Redo the previous query, but retrieve only the Employee ID, Employee Name and Job Title.

Solution

```
select EMP_ID,F_NAME,L_NAME,  JOB_TITLE from employees E, jobs J where E.JOB_ID = J.JOB_IDENT;
```

Output

Result set 1

Search

EMP_ID	F_NAME	L_NAME	JOB_TITLE
E1001	John	Thomas	Sr. Architect
E1002	Alice	James	Sr.Software Dev
E1003	Steve	Wells	Jr.Software Dev
E1004	Santosh	Kumar	Jr.Software Dev
E1005	Ahmed	Hussain	Jr. Architect
E1006	Nancy	Allen	Lead Architect
E1007	Mary	Thomas	Jr. Designer
E1008	Bharath	Gupta	Jr. Designer
E1009	Andrea	Jones	Sr. Designer
E1010	Ann	Jacob	Sr. Designer

5. Problem:

Redo the previous query, but specify the fully qualified column names with aliases in the SELECT clause.

Solution

```
select E.EMP_ID,E.F_NAME,E.L_NAME,  J.JOB_TITLE from employees E, jobs J where E.JOB_ID = J.JOB_IDENT;
```

Output

Result set 1

Search

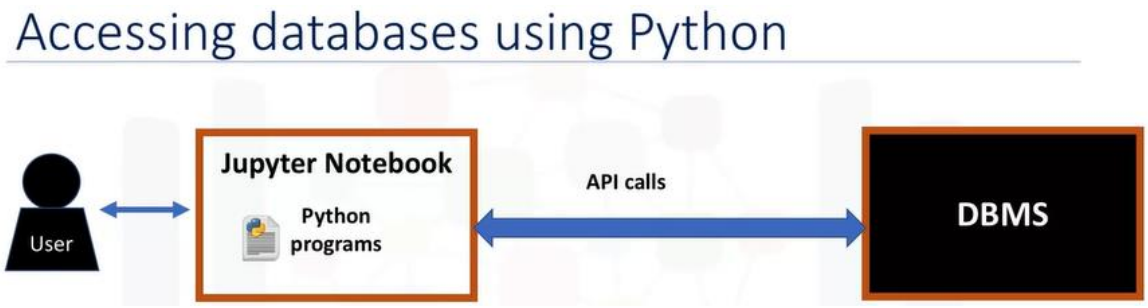
EMP_ID	F_NAME	L_NAME	JOB_TITLE
E1001	John	Thomas	Sr. Architect
E1002	Alice	James	Sr.Software Dev
E1003	Steve	Wells	Jr.Software Dev
E1004	Santosh	Kumar	Jr.Software Dev
E1005	Ahmed	Hussain	Jr. Architect
E1006	Nancy	Allen	Lead Architect
E1007	Mary	Thomas	Jr. Designer
E1008	Bharath	Gupta	Jr. Designer
E1009	Andrea	Jones	Sr. Designer
E1010	Ann	Jacob	Sr. Designer

Solution Script

If you would like to run all the solution queries of the SQL problems of this lab with a script, download the script below. Upload the script to the Db2 console and run. Follow Hands-on Lab : Create tables using SQL scripts and Load data into tables on how to upload a script to Db2 console and run it.

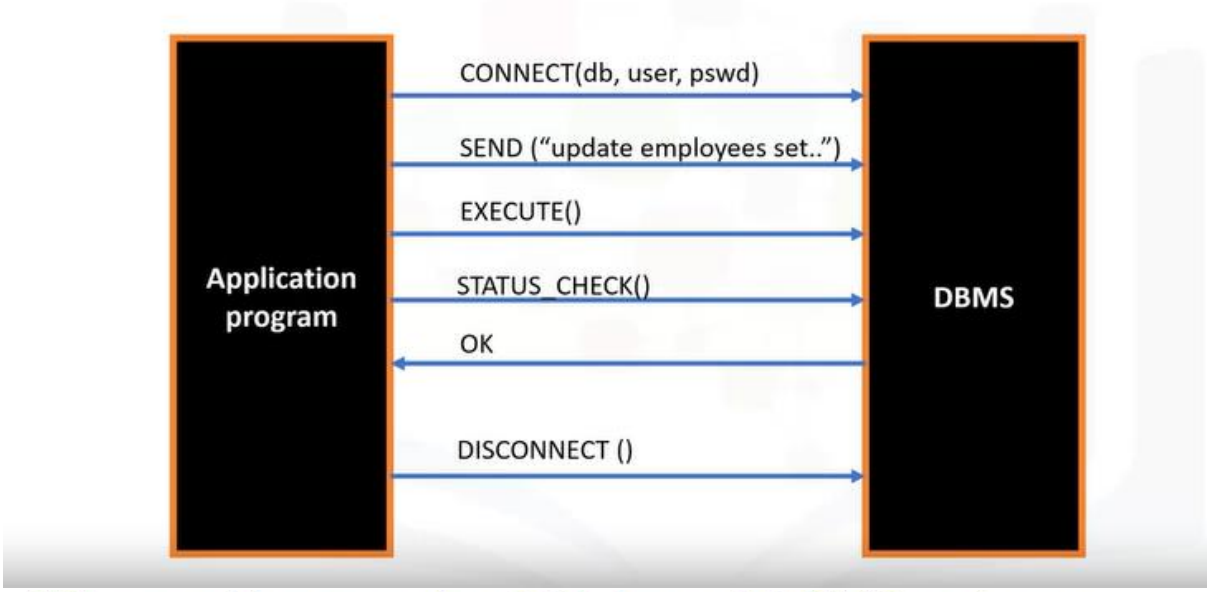
- MultipleTables_Solution_Script.sql

How to Access Databases Using Python



An application programming interface is a set of functions that you can call to get access to some type of service. The SQL API consists of library function calls as an application programming interface, API, for the DBMS. To pass SQL statements to the DBMS, an application program calls functions in the API, and it calls other functions to retrieve query results and status information from the DBMS. The basic operation of a typical SQL API is illustrated in the figure. The application program begins its database access with one or more API calls that connect the program to the DBMS. To send the SQL statement to the DBMS, the program builds the statement as a text string in a buffer and then makes an API call to pass the buffer contents to the DBMS. The application program makes API calls to check the status of its DBMS request and to handle errors. The application program ends its database access with an API call that disconnects it from the database

What is a SQL API?

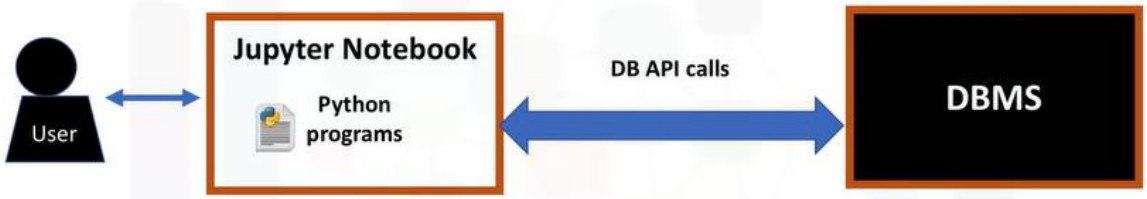


APIs used by popular SQL-based DBMS systems

Application or Database	SQL API
MySQL	MySQL C API
PostgreSQL	psycopg2
IBM DB2	ibm_db
SQL Server	dblib API
Database access for Microsoft Windows OS	ODBC
Oracle	OCI
Java	JDBC

Writing code using DB-API

What is a DB-API?



- Python's standard API for accessing relational databases
- Allows a single program that to work with multiple kinds of relational databases
- Learn DB-API functions once, use them with any database

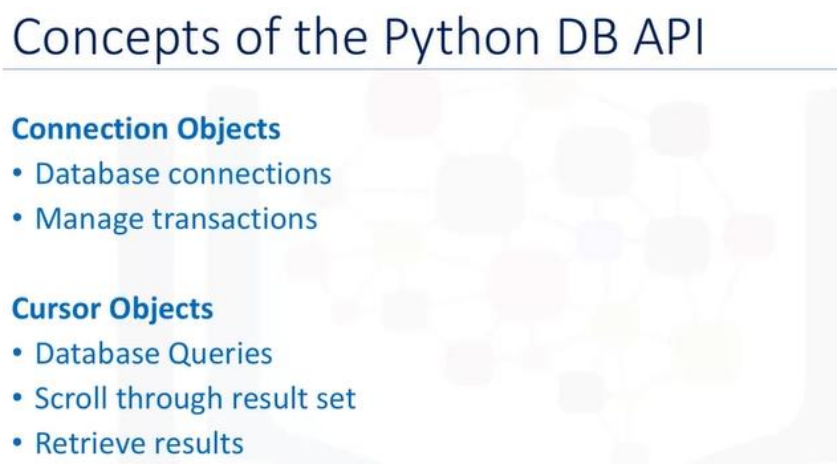
Benefits of using DB-API

- Easy to implement and understand
- Encourages similarity between the Python modules used to access databases
- Achieves consistency
- Portable across databases
- Broad reach of database connectivity from Python

Examples of libraries used by database systems to connect to Python applications

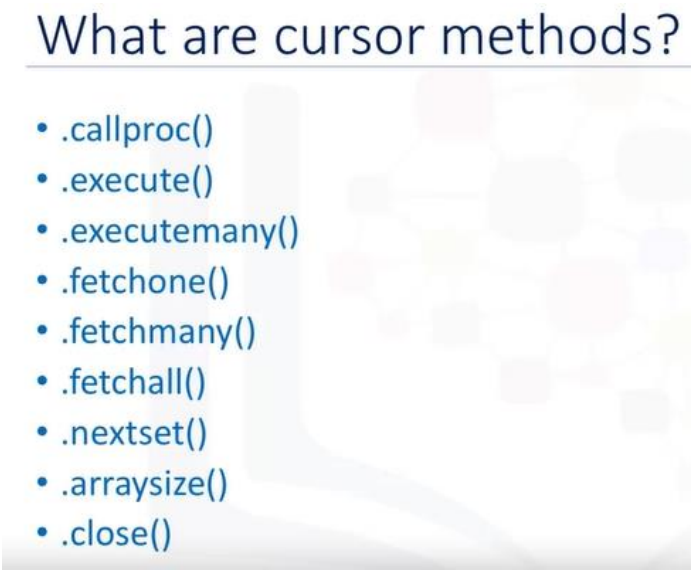
Database	DB API
IBM Db2	Ibm_db
Compose for MySQL	MySQL Connector/Python
Compose for PostgreSQL	psycopg2
Compose for MongoDB	PyMongo

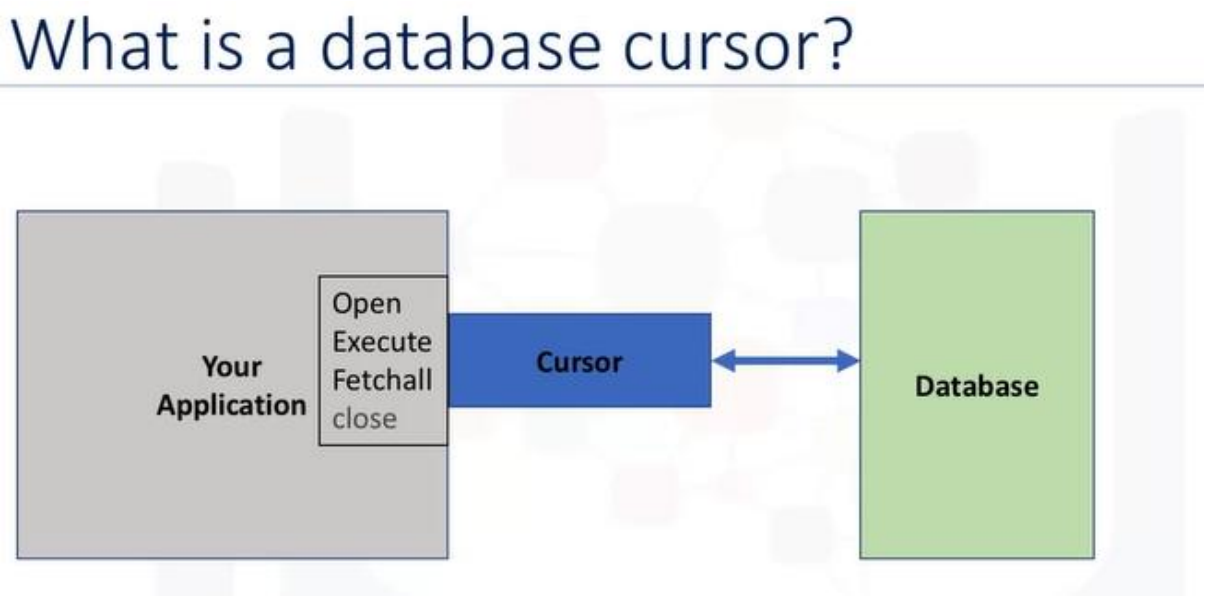
The two main concepts in the Python DB-API are connection objects and query objects. You use connection objects to connect to a database and manage your transactions. Cursor objects are used to run queries. You open a cursor object and then run queries. The cursor works similar to a cursor in a text processing system where you scroll down in your result set and get your data into the application. Cursors are used to scan through the results of a database. The DB_API includes a connect constructor for creating a connection to the database. It returns a Connection Object, which is then used by the various connection methods.



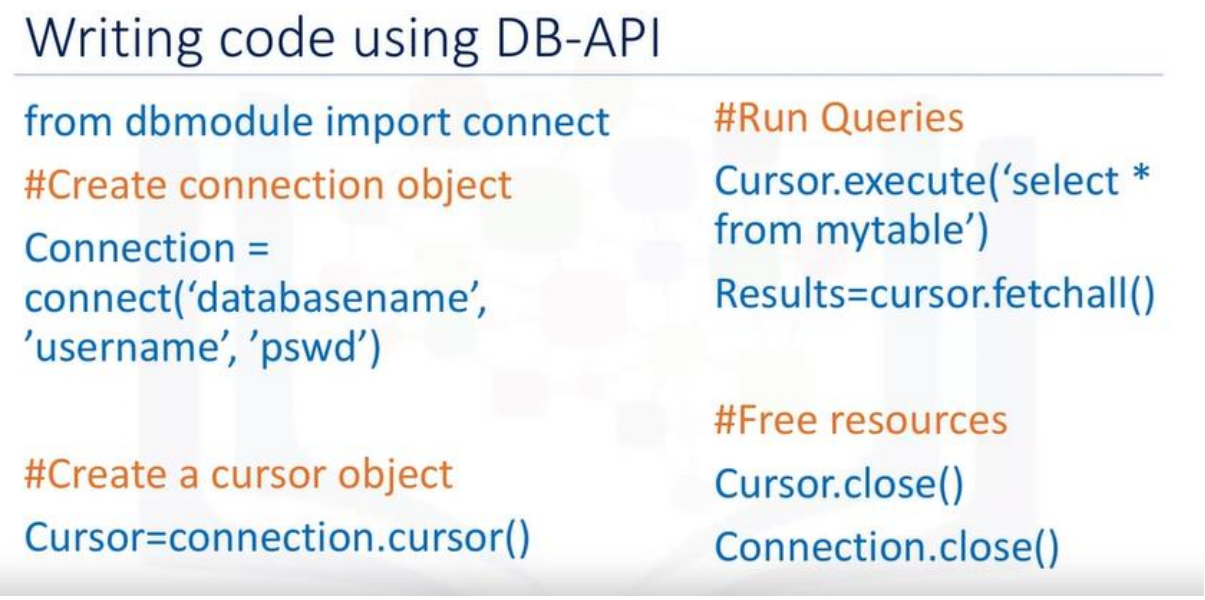
These connection methods are: The cursor() method, which returns a new cursor object using the connection. The commit() method, which is used to commit any pending transaction to the database. The rollback() method, which causes the database to roll back to the start of any pending transaction. The close() method, which is used to close a database connection.

These objects represent a database cursor, which is used to manage the content of a fetch operation. Cursors created from the same connection are not isolated that is, any changes done to the database by a cursor are immediately visible by the other cursors. Cursors created from different connections can or cannot be isolated depending on how the transaction support is implemented.



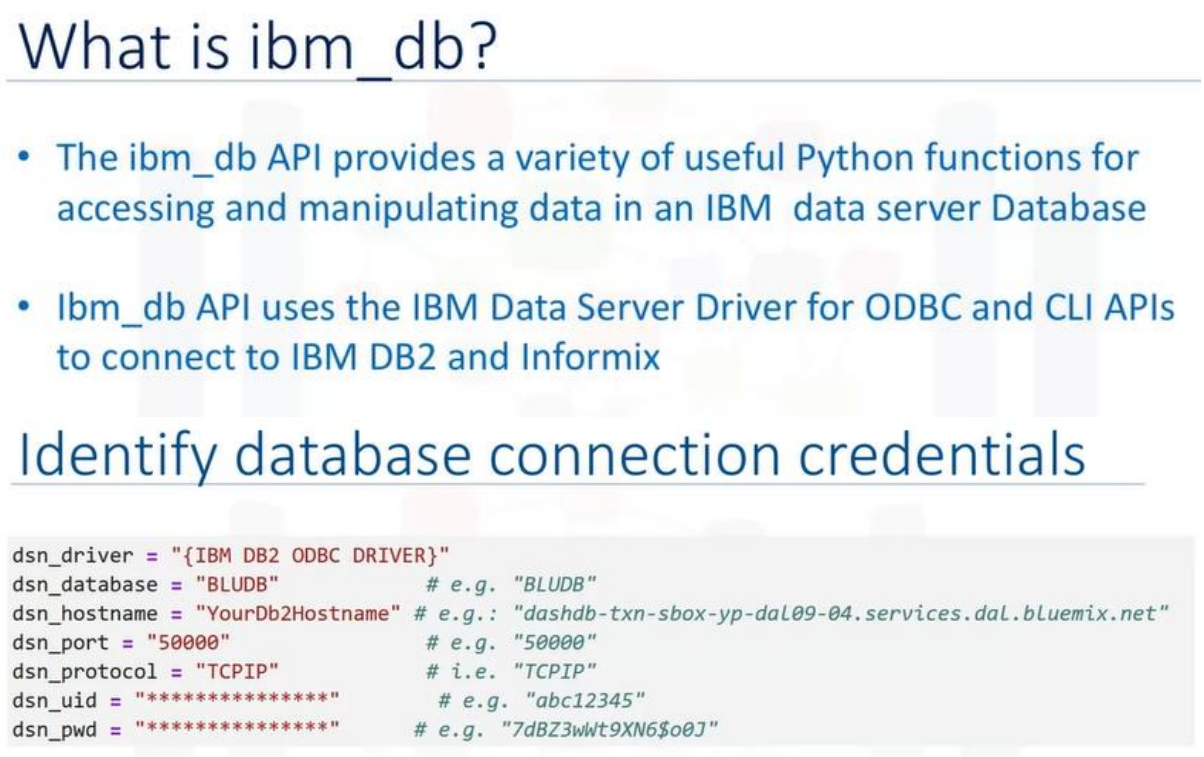


A database cursor is a control structure that enables traversal over the records in a database. It behaves like a file name or file handle in a programming language. Just as a program opens a file to access its contents, it opens a cursor to gain access to the query results. Similarly, the program closes a file to end its access and closes a cursor to end access to the query results. Another similarity is that just as file handle keeps track of the program's current position within an open file, a cursor keeps track of the program's current position within the query results.



First, you import your database module by using the connect API from that module. To open a connection to the database, you use the connect constructor and pass in the parameters, that is, the database name, username, and password. The connect function returns connection object. After this, you create a cursor object on the connection object. The cursor is used to run queries and fetch results. After running the queries, using the cursor, we also use the cursor to fetch the results of the query. Finally, when the system is done running the queries, it frees all resources by closing the connection. Remember that it is always important to close connections to avoid unused connections taking up resources.

Connecting to a database using ibm_db API



Connecting to the DB2 requires the following information: a driver name, a database name, a host DNS name or IP address, a host port, a connection protocol, a user ID, and a user password.

Create a database connection

```
#Create database connection
dsn = (
    "DRIVER={{IBM DB2 ODBC DRIVER}};"
    "DATABASE={0};"
    "HOSTNAME={1};"
    "PORT={2};"
    "PROTOCOL=TCPIP;"
    "UID={3};"
    "PWD={4};").format(dsn_database, dsn_hostname, dsn_port, dsn_uid, dsn_pwd)

try:
    conn = ibm_db.connect(dsn, "", "")
    print ("Connected!")

except:
    print ("Unable to connect to database")
```

```
Connected!
```

Close the database connection

```
In [8]: ibm_db.close(conn)

Out[8]: True
```

Create credentials to access your database instance

Exercise

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.

1. Go to your IBM Cloud Resources dashboard (or click on IBM Cloud in the top left corner):

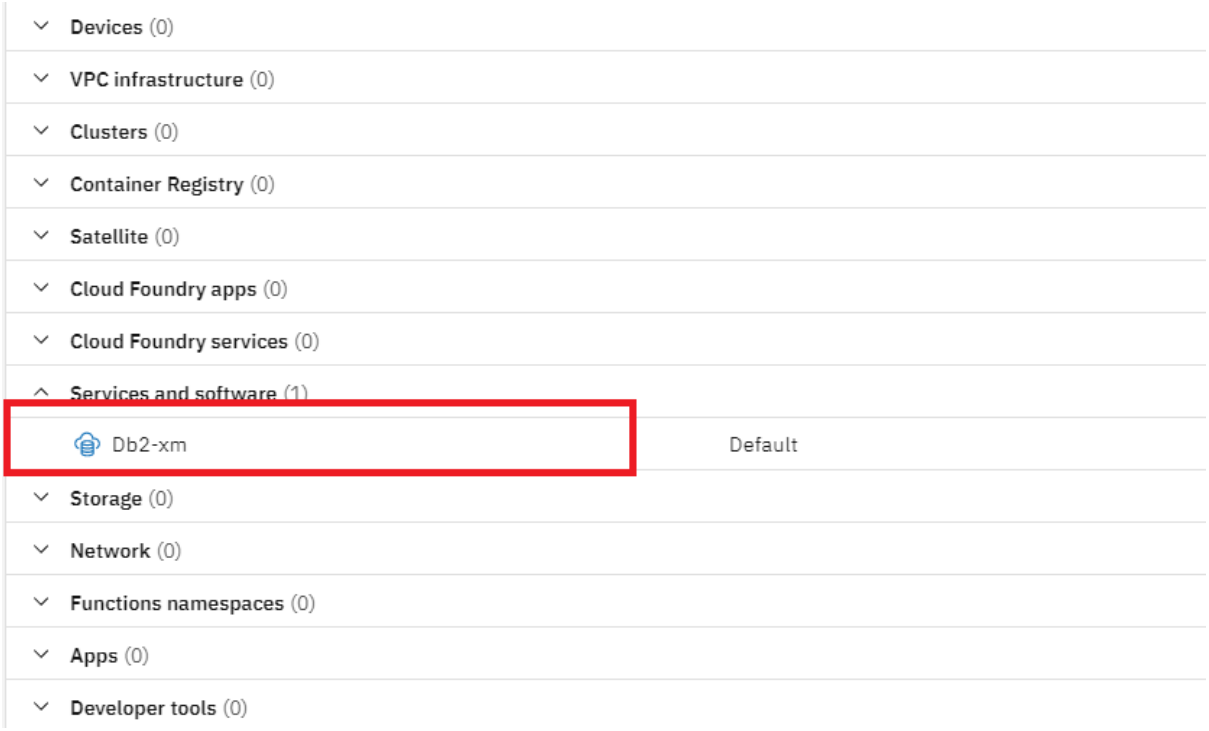
https://cloud.ibm.com/resources

Note: you may need to log into IBM Cloud in the process of loading the resources/dashboard.

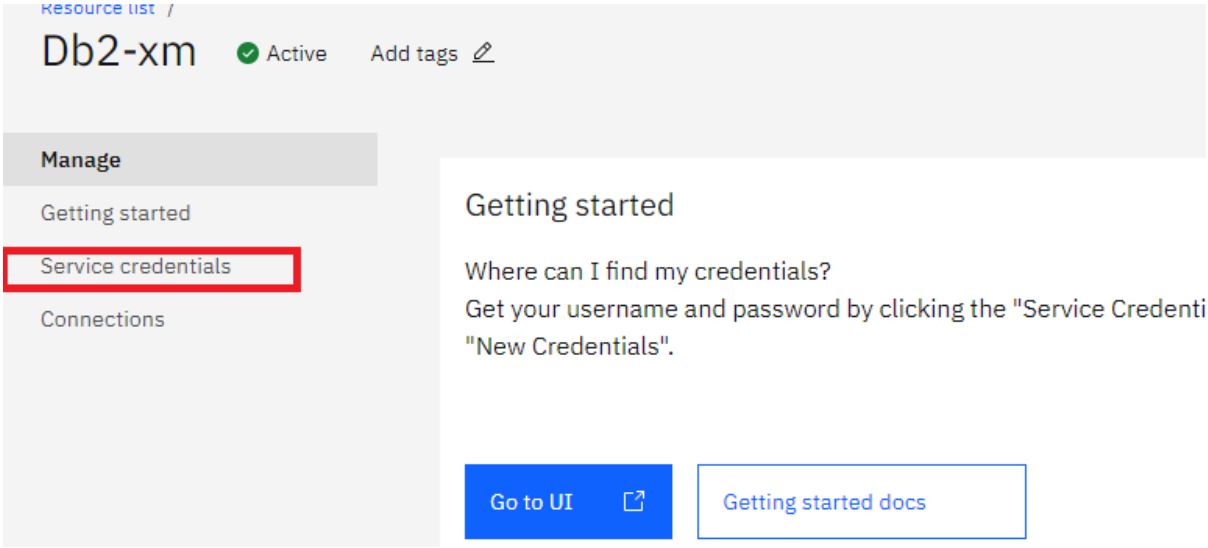
If your connection is slow it may take over 30 seconds for the dashboard to fully load.

1. Locate and click on your Db2 service listed under Services.

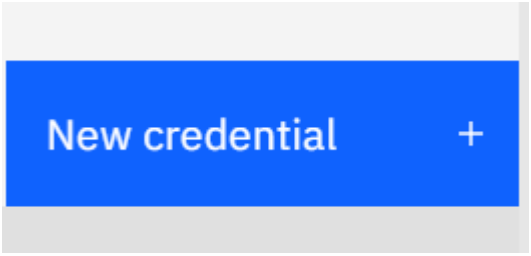
(**NOTE:** In the example below the service is called "Db2-xx" but your Db2 service may have a different letters/number in the suffix e.g. "Db2-f8", "Db-50", etc.)



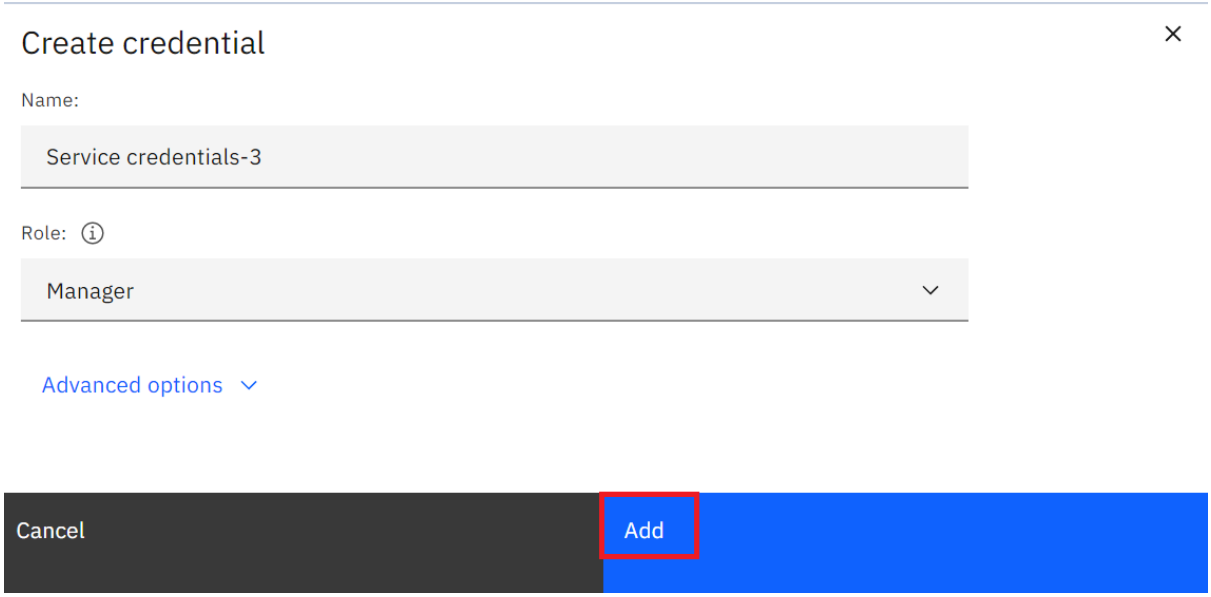
2. Click on Service Credentials in the left menu



3. Click on the button to create **New credential**

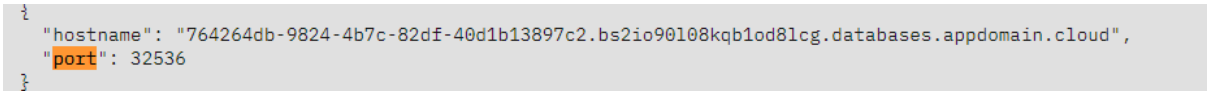


In the prompt that comes up click the "Add" button in the bottom right:



4. Check the box to **View credentials**
5. Copy and save the credentials making a note of the following:

- **port** is the database port



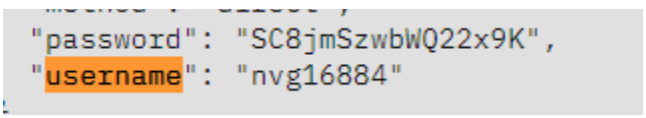
"hostname": "764264db-9824-4b7c-82df-40d1b13897c2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud",

"port": 32536

- **db** is the database name
- **host** is the hostname of the database instance

Host: "764264db-9824-4b7c-82df-40d1b13897c2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:32536"

- **username** is the username you'll use to connect
- **password** is the password you'll use to connect



"password": "SC8jmSzwbWQ22x9K",

"username": "nvg16884"

```
Service credentials-1                2021-09-27 11:07 AM
{
  "connection": {
    "cli": {
      "arguments": [
        [
          "-u",
          "nvg16884",
          "-p",
          "SC8jmSzwbWQ22x9K",
          "--ssl",
          "--sslCAFile",
          "1dd14d0c-1b52-4f63-a606-53ecba28771d",
          "--authenticationDatabase",
          "admin",
          "--host",
          "764264db-9824-4b7c-82df-40d1b13897c2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:32536"
        ]
      ],
      "bin": "db2",
      "certificate": {
        "certificate_base64": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURIVENDQWdXZ0F3SUJBZ0lVT3dvMC9va09CUEN5RjFWeFJxVGhKRW9ubDBVd0RRWUpLb1pJaHZjTkFRRUwKQlFBd0hqRWNN
Qm9HQTFVRUF3d1RTVUpOSUVOc2IzVmtJRVJoZEdGaVlYTmxjekFlRncweU1EQTRNRFF3TWpVMwpNalphRncwek1EQTRNREl3TWpVM01qWmFNQjR4SERBYUJnTlZCQU1NRTBsQ1RQkRiRzkxWkNRVlYUmhZbUZ6ClpYTXd
nZ0VpTUEwR0NTcUdTSWIzRFFFQkFRVUFBNElCRHdBd2dnRUtBb0lCQVFEb0ZFNGQ0SGdOeXZMUVIwR3gKQTB0amRXQnM4NVBjTDNyRStjN1R3K2diRUdQSUxJU0VZV3o4Y1g1TG1XQk0rY1FnOG9VeSsrQXJ3OEoxaXdRZQ
pySmlIU2I1clF4WTM0c3BQeGRFVEZkWEhScnJhMGU2VmM4MW42TllJL0ZHSnl1Q3hrTG5GMUtFQW9hbHYwaDM2CnhDT0FvcXRwTlFrTzNpMTRGeU0yRDRiajkxckI4RGk4Vy9XMVpVdVhMNGwzZXVLZUVCeTRuZmhJV3kyS
Vc3aUMKbGpMZ3RlN3hZTDVHbVpKOUdsYWtrSnJ1cnpNREFQLzVUYnRlUUIydElodTBRSVRFZHlESVFYUEZGRDBHYzloZAo3M29JdnpVZUJ3VC9uRHN3OTJNNC82SkdtZWpKN0lpdFBTN3Y2a2dlUVhIND1BaUVJNXpQdUVp
VzNOYi9GR0pYCmY2a2JBZ01CQUFHalV6Q1JNQjBHQTFVZERnUVdCQlR2RzZ2RU5MRjFVbWZnQ003MmxOcmMzSDI2bURBZkJnTlYKSFNNRUdEQVdnQlR2RzZ2RU5MRjFVbWZnQ003MmxOcmMzSDI2bURBUEJnTlZIUk1CQWY
```


You need to scroll down to get the credentials details.

```
plset"
        ],
        "database": "bludb",
        "host_ros": [
          "764264db-9824-4b7c-82df-40d1b13897c2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:32761"
        ],
        "hosts": [
          {
            "hostname": "764264db-9824-4b7c-82df-40d1b13897c2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud",
            "port": 32536
```

Hands-on Lab: Connecting to a database instance

This Lab will demonstrate how to create a database connection to an instance of IBM Db2 on Cloud. You will be using Jupyter notebooks to complete the labs. Click HERE to download the lab notebook (.ipynb)

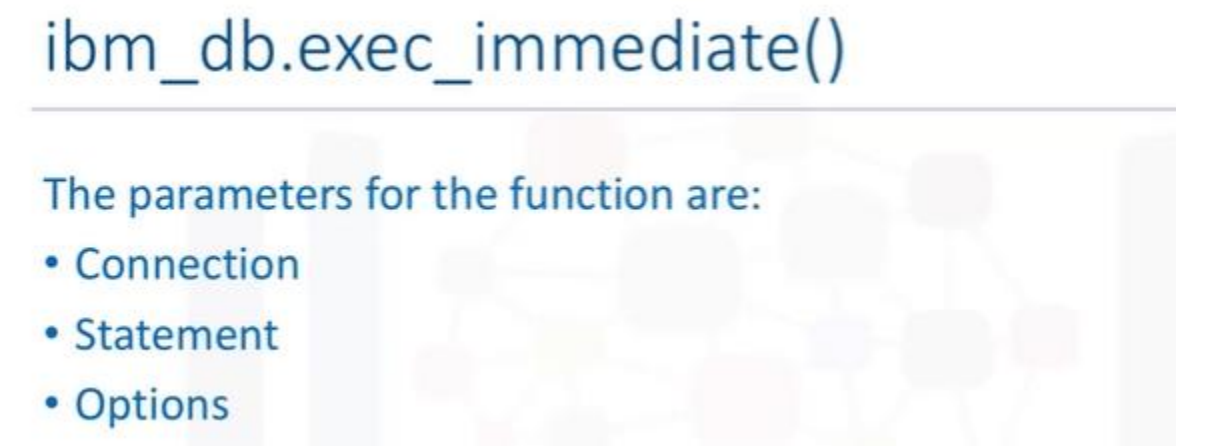
Local file can be downloaded from here → Python File

Assignment can be seen from here → Python File

Creating tables, loading data and querying data

We will be using DB2 as the database.

Let's see how we can create the Trucks table in the DB2 using Python code.



To create a table, we use the ibm_db.exec_immediate function. The parameters for the function are connection, which is a valid database connection resource that is returned from the ibm_db.connect or ibm_db.pconnect function statement, which is a string that contains the SQL statement, and options which is an optional parameter that includes a dictionary that specifies the type of cursor to return for results sets.

Python code to create a table

```
stmt = ibm_db.exec_immediate(conn,
"CREATE TABLE Trucks(
serial_no varchar(20) PRIMARY KEY NOT NULL,
model VARCHAR(20) NOT NULL,
manufacturer VARCHAR(20) NOT NULL,
Engine_size VARCHAR(20) NOT NULL,
Truck_Class VARCHAR(20) NOT NULL) "
)
```

Python code to insert data into the table

```
stmt = ibm_db.exec_immediate(conn,
"INSERT INTO Trucks(serial_no,
model,manufacturer,Engine_size, Truck_Class)
VALUES('A1234','Lonestar','International
Trucks','Cummins ISX15','Class 8');")
```

Insert more rows to the table

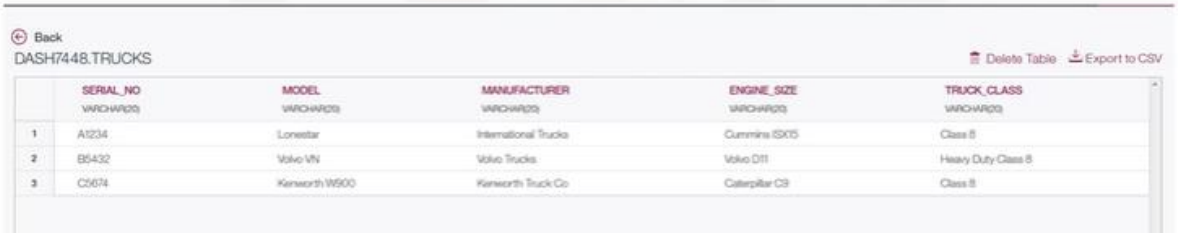
```
stmt = ibm_db.exec_immediate(conn,
"INSERT INTO Trucks(serial_no,model,manufacturer,Engine_size, Truck_Class)
VALUES('B5432','Volvo VN','Volvo Trucks','Volvo D11','Heavy Duty Class 8');")
```

```
stmt = ibm_db.exec_immediate(conn,
"INSERT INTO Trucks(serial_no,model,manufacturer,Engine_size, Truck_Class)
VALUES('C5674','Kenworth W900','Kenworth Truck Co','Caterpillar C9','Class 8');")
```

Python code to query data

```
In [10]: stmt = ibm_db.exec_immediate(conn,"SELECT * FROM Trucks")

         ibm_db.fetch_both(stmt)

Out[10]: {0: 'A1234',
          1: 'Lonestar',
          'MANUFACTURER': 'International Trucks',
          3: 'Cummins ISX15',
          'SERIAL_NO': 'A1234',
          'ENGINE_SIZE': 'Cummins ISX15',
          'MODEL': 'Lonestar',
          'TRUCK_CLASS': 'Class 8',
          2: 'International Trucks',
          4: 'Class 8'}
```

Confirm the output on Db2 on Cloud console



Using pandas

```
In [19]: import pandas
         import ibm_db_dbi
         pconn = ibm_db_dbi.Connection(conn)
         df = pandas.read_sql('SELECT * FROM Trucks', pconn)
         df
```

Out[19]:

	SERIAL_NO	MODEL	MANUFACTURER	ENGINE_SIZE	TRUCK_CLASS
0	A1234	Lonestar	International Trucks	Cummins ISX15	Class 8
1	B5432	Volvo VN	Volvo Trucks	Volvo D11	Heavy Duty Class 8
2	C5674	Kenworth W900	Kenworth Truck Co	Caterpillar C9	Class 8

Hands-on Lab: Creating tables, inserting and querying Data

Lab Notebook

Click HERE to download the lab notebook (.ipynb)

Introducing SQL Magic

Objective

In this reading, you will learn about the SQL magic commands.

Jupyter notebooks have a concept of **Magic** commands that can simplify working with Python, and are particularly useful for data analysis. Your notebooks can have two types of magic commands:

Cell magics: start with a double **%%** sign and apply to the entire cell

Line magics: start with a single **%** (percent) sign and apply to a particular line in a cell

Their usage is of the format:

%magicname arguments

So far in the course you learned to accessed data from a database using the Python DB-API (and specifically ibm_db). With this API execution of queries and fetching their results involves multiple steps. You can use the **SQL Magic** commands to execute queries more easily.

For example if you want to execute the a query to select some data from a table and fetch its results, you can simply enter a command like the following in your Jupyter notebook cell:

%sql select * from tablename

Although SQL magic simplifies working with databases, it has some limitations. For example, unlike DB-API, there are no explicit methods to close a connection and free up resources.

Hands-on Tutorial: Accessing Databases with SQL magic

In this hands-on tutorial, you will create a table, insert some data, and retrieve the results using SQL magic. Click HERE to download the lab notebook (.ipynb) or Lab Notebook