Welcome to SQL for Data Science

The demand for data scientists is high, boasting a median base salary of $110,000 and job satisfaction score of 4.4 out of five.

It's no wonder that it's the top spot on Glassdoor's best jobs in America. Glassdoor analyzed data from data scientist job postings on Glassdoor and found that SQL is listed as one of the top three skills for a data scientist.

Before you step into the field of data science,

it is vitally important that you set yourself apart by mastering the foundations of this field.

One of the foundational skills that you will require is SQL.

SQL is a powerful language that's used for communicating with databases.

- Every application that manipulates any kind of data needs to store that data somewhere; whether it's big data, or just a table with a few simple rows for government, or a small startup, or a big database that spans over multiple servers or a mobile phone that runs its own small database.

Here are some of the advantages of learning SQL for someone interested in data science.

- SQL will boost your professional profile as a data scientist, as it is one of the most sought after skills by hiring employers.

- Learning SQL will give you a good understanding of relational databases.
- Tapping into all this information requires being able to communicate with the databases that store the data.
    - Even if you work with reporting tools that generate SQL queries for you, it may be useful to write your own SQL statements so that you need not wait for other team members to create SQL statements for you.

In this course, you will learn the basics of both the SQL language and relational databases.

The course includes interesting quizzes and hands on lab assignments, where you can get experience working with databases.

In the first few modules, you work directly with the database and develop a working knowledge of SQL. Then, you will connect to a database and run

SQL queries like a data scientist typically would, where you will use Python and Jupyter notebooks to connect to relational databases to access and analyze data.

There is also an assignment included towards the end of the course,

where you will get an opportunity to apply the concepts that you learned.

So, let's get started with SQL for data science.

Why SQL for Data Science

Why SQL for Data Science

- Median based salary: $110,000
- Job satisfaction score: 4.4/5
- Top spot on Glassoor's best jobs in America
- Top three skills for a Data Scientist

Why SQL for Data Science

- Big data
- Table with a few rows
- Small start up
- Big Database
- Mobile phone

Why SQL for Data Science

Advantages:

- Boost your professional profile
- Give you a good understanding of relational databases

Course details

- Basics for SQL and Relational Databases
- Working knowledge of SQL and Databases
- Connect to Database and run SQL queries
- Python and Jupyter Notebooks to Analyze Data
- Assignment to apply concepts with Real World Dataset

Course Overview

Bookmarked

Course Overview

Much of the world's data resides in databases. SQL (or Structured Query Language) is a powerful language which is used for communicating with and extracting data from databases. A working knowledge of databases and SQL is a must if you want to become a data scientist.

The purpose of this course is to introduce relational database concepts and help you learn and apply foundational knowledge of the SQL language. It is also intended to get you started with performing SQL access in a data science environment.

The emphasis in this course is on hands-on and practical learning. As such, you will work with real databases, real data science tools, and real-world datasets. You will create a database instance in the cloud. Through a series of hands-on labs, you will practice building and running SQL queries. You will also learn how to access databases from Jupyter notebooks using SQL and Python.

Who Should Take This Course

Bookmarked

Who should take this course?

This course is intended for existing or aspiring Data Scientists who want practical knowledge of SQL to query databases and know-how to execute SQL from Jupyter Notebooks using Python. It is also useful for Data Analysts, Application Developers, and Data Engineers.

Prerequisites

Bookmarked

Pre-requisites

There are no prerequisites for this course other than general familiarity with using computers and a desire to learn. No prior knowledge of databases, SQL, or programming is required. Some experience with Python will be an asset, it is not a must, as we will share the Python working knowledge required for completing this course.

Changelog

Bookmarked

Change Log

01 Sept 2020 (Rav Ahuja): Updated version of the course published on edX.org.

01 Sept 2020 (Sonali Gupta): Replaced links to labs with links from SN Asset Library.

17 March 2020 (Rav Ahuja): Bonus (Optional) Module on JOINs added at the end of the course.

15 March 2020 (Rav Ahuja): Assessment for Final Assignment added.

14 March 2020 (Rav Ahuja): Added Jupyter Lab for final assignment. Bug fixes: Resolved issues with launching Practice Lab in Module 6.

13 March 2020 (Rav Ahuja): Bug fixes: Fixed Broken Links in Module 3 Lab. Fixed issues with Module 4 Quiz.

9 March 2020 (Rav Ahuja): Initial version of the course published on edX.org

Learning Objectives

Bookmark this page

Learning Objectives

In this course you will learn about:

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

Syllabus

Bookmarked

Syllabus

Module 1 - Getting Started with SQL

- Introduction to Databases
- SELECT Statement
- SELECT Statement Examples
- Hands-on Lab: Simple SELECT Statements
- COUNT, DISTINCT, LIMIT
- Hands-on Lab: COUNT, DISTINCT, LIMIT
- INSERT Statement
- UPDATE and DELETE Statements
- Summary & Highlights
- Practice Quiz
- Graded Quiz

Module 2 - Introduction to Relational Databases and Tables

- Relational Database Concepts
- How to Create a Database Instance on Cloud
- Hands-on Lab: Sign up for IBM Cloud, Create Db2 Service Instance and Get started with the Db2 Console
- Types of SQL Statements (DDL vs. DML)
- CREATE TABLE Statement
- ALTER, DROP and Truncate Tables
- Examples to CREATE and DROP Tables
- Hands-on Lab: CREATE, ALTER, TRUNCATE, DROP
- Hands-on Lab: Create and Load Tables using SQL Scripts
- Summary & Highlights
- Practice Quiz
- Graded Quiz

Module 3 - Intermediate SQL

- Using String Patterns and Ranges
- Sorting Result Sets
- Grouping Result Sets
- Hands-on Lab: String Patterns, Sorting & Grouping
- Summary & Highlights
- Practice Quiz
- Built-in Database Functions
- Date and Time Built-in Functions
- Hands-on Lab: Built-in Functions
- Sub-Queries and Nested Selects
- Hands-on Lab: Sub-Queries and Nested SELECTs
- Working with Multiple Tables
- Hands-on Lab: Working with Multiple Tables
- Summary & Highlights
- Practice Quiz
- Graded Quiz

Module 4 - Accessing Databases using Python

- How to Access Databases Using Python
- Writing Code Using DB-API
- Connecting to a Database Using ibm_db API
- Hands-on Lab: Create Database Credentials
- Hands-on Lab: Connecting to a Database Instance
- Creating Tables, Loading Data, and Querying Data
- Hands-on Lab: Creating Tables, Inserting and Querying Data
- Introducing SQL Magic
- Hands-on Lab: Analyzing a Real World Data Set
- Summary & Highlights
- Practice Quiz
- Graded Quiz

Module 5 - Final Exam

- Instructions
- Final Exam

Module 6 - Assignment Preparation: Working with real-world data sets and built-in SQL functions

- Working with Real World Datasets
- Getting Table and Column Details
- LOADing Data
- Hands-on Lab: Practice Querying Real World Datasets

Module 7 - Course Assignment

Instructions for Peer-Graded Assignment

Jupyter Notebook with Problems for Peer-Reviewed Assignment

Peer Review: Submit Your Work and Review Your Peers

Grading Scheme

Bookmarked

GRADING SCHEME

This section contains information for those earning a certificate. Those auditing the course can skip this section and click next.

1. The course contains 5 Graded Quizzes (1 per module) worth 5o% of the total grade. Each Graded Quiz carries an equal weight of 10% of the total grade.
2. The course also includes a Final Exam worth 25% of the total grade.
3. The course also includes a Final Assignment worth 25% of the total grade.
4. The minimum passing mark for the **course** is 70%.
5. Permitted attempts are per **question**:

Graded Quizzes:

- One attempt - For True/False questions
- Two attempts - For any question other than True/False

Final Assignment:

- One attempt - For ALL questions

5. When there is more than 1 attempt permitted, there are no penalties for incorrect attempts.

6. Clicking the "**Submit**" button when it appears, means your submission is **FINAL**. You will **NOT** be able to resubmit your answer for that question again.

7. Check your grades in the course at any time by clicking on the "Progress" tab.

Module Introduction & Learning Objectives

Bookmark this page

In this module, you will learn some basic SQL statements and practice them hands-on on a live database.

Learning Objectives

- Describe SQL and Databases
- Explain the syntax of basic SQL statements - Select, Insert, Update, Delete
- Compose and execute basic SQL statements hands-on on a live database
- Demonstrate how to write basic SQL statements

Introduction to Databases

Hello and welcome to SQL for data science.

First, we will talk a little bit about what you'll learn in this course.

This course teaches you the basics of the SQL language and the relational database model.

There will be some lab exercises, and at the end of each section there are a few review questions.

And at the end, there is a final exam. By the end of this course, you will be able to discuss SQL basics and explain various aspects of the relational database model.

In this video, we will learn about SQL and relational databases. By the end of this video,

you will be able to describe SQL, data, database, a relational database, and list five basic SQL commands. But wait, what is SQL and what is a relational database? What is SQL?

SQL is

- a language used for relational databases to query or get data out of a database.
- SQL is also referred to as SQL and is short for its original name Structured English Query Language.
- SQL is a language used for a database to query data.

But what is data and what is a database?

Data

- is a collection of facts in the form of words, numbers, or even pictures.
- Data is one of the most critical assets of any business.

It is used and collected practically everywhere. Your bank stores data about you, your name,

address, phone number, account number et cetera.Your credit card company and your paypal accounts also store data about you.

Data is important; so, it needs to be secure, and it needs to be stored and accessed quickly.

The answer is a database.

So, what is a database?

Databases are everywhere and used every day, but they are largely taken for granted.

- A database is a repository of data.
- It is a program that stores data.
- A database also provides the functionality for adding, modifying, and querying that data.

There are different kinds of databases of different requirements.

The data can be stored in various forms.

When data is stored in tabular form,

the data is organized in tables like in a spreadsheet, which is columns and rows.

That's a relational database.

The columns contain properties about the item such as last name, first name, email address, city.

A table is a collection of related things like a list of employees or a list of book authors.

In a relational database, you can form relationships between tables.

So a database is a repository of data.

A set of software tools for the data in the database is called a database management system or DBMS for short.

The terms database, database server, database system, data server, and database management systems are often used interchangeably.

For relational databases, it's called a relational database management system or RDBMS.

- RDBMS is a set of software tools that controls the data such as access, organization, and storage.
- And RDBMS serves as the backbone of applications in many industries including banking,

transportation, health, and so on.

Examples of relational database management systems are my SQL, Oracle Database,

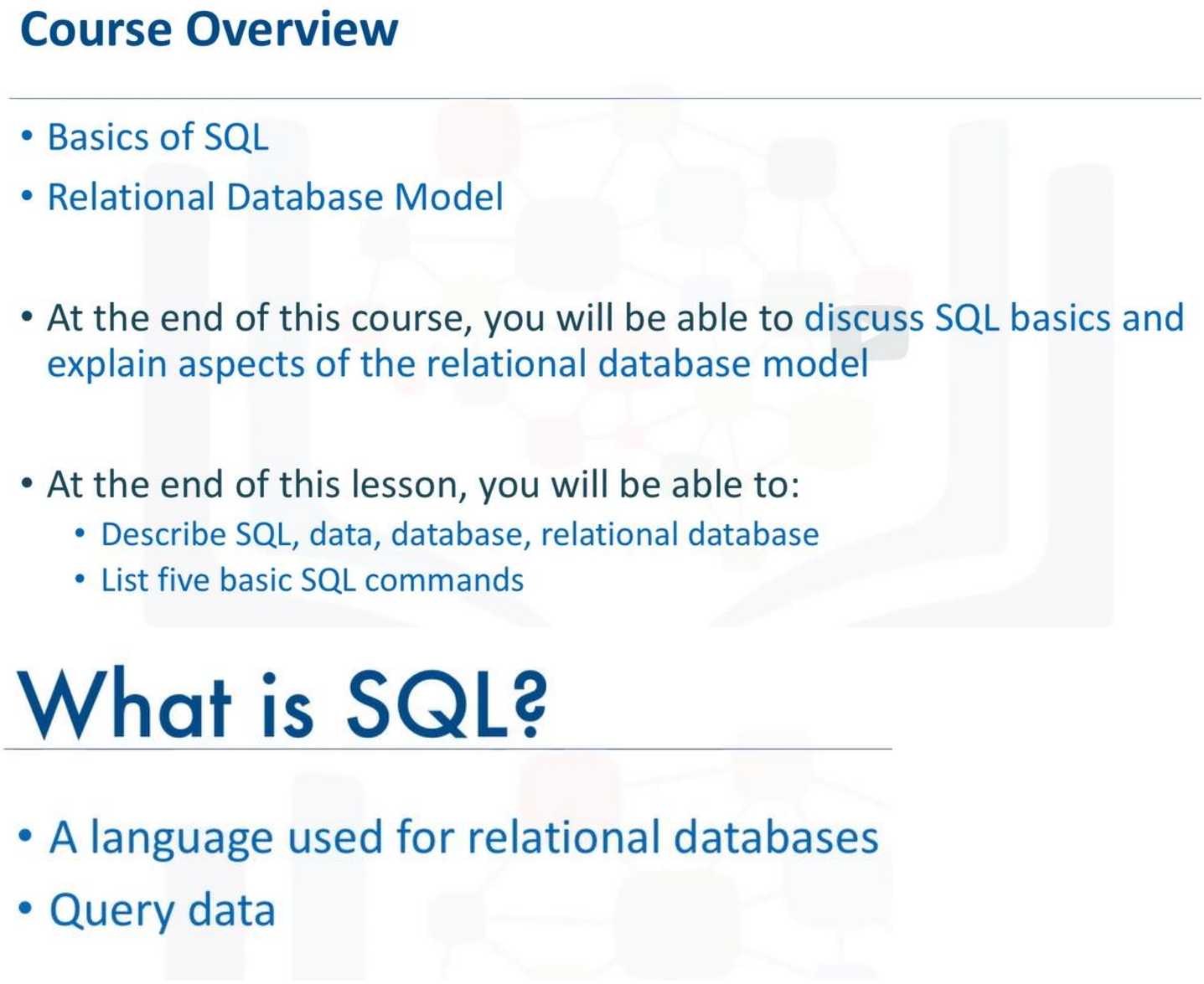
DB2 Warehouse, and DB2 on Cloud.

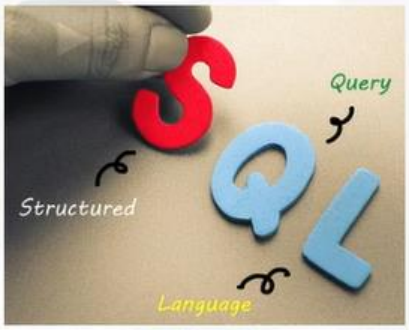
For the majority of people using a database, there are five simple commands to create a table, insert data to populate the table, select data from the table, update data in the table, delete data from the table.

So those are the building blocks for SQL for data science.

You can now describe what is SQL, what is data, what is a database, and what is a relational database. You know that RDBMS stands for Relational Database Management System,

and you can list five basic SQL commands to create a table, insert data to populate the table, select data from the table, update data in the table, and delete data from the table.





What is data?

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

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

Relational Database

- Data stored in tabular form - columns and rows
- Columns contain item properties e.g. Last Name, First Name, etc.
- Table is collection of related things e.g. Employees, Authors, etc.
- Relationships can exist between tables (hence: "relational")

Student ID	First Name	Last Name
34933	Victoria	Slater
93759	Justin	McNeil
20847	Jessica	Bennett
65947	Michelle	Dolin
24956	David	Price
65692	Franklin	Mullins
24271	Alissa	Lee

DBMS

- Database: repository of data
- DBMS: Database Management System - software to manage databases
- Database, Database Server, Database System, Data Server, DBMS - often used interchangeably

What is RDBMS?

- RDBMS = Relational database management system
- A set of software tools that controls the data
  - access, organization, and storage
- Examples are: MySQL, Oracle Database, IBM Db2, etc.

Basic SQL Commands

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

Summary

- You can now describe:
  - SQL
  - Data
  - Database
  - Relational Databases
  - RDBMS
  - 5 basic SQL commands:
    - Create, Insert, Select, Update, Delete

SELECT Statement

Hello and welcome to retrieving data with a SELECT statement. In this video, we will learn about retrieving data from a relational database table by selecting columns of a table.

At the end of this lesson, you will be able to retrieve data from a relational database table,

to find the use of a predicate, identify the syntax of the SELECT statement using the WHERE clause, and list the comparison operators supported by a relational database management system.

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

So, after creating a relational database table and inserting data into the table, we want to see the data.

To see the data, we use the SELECT statement.

The SELECT statement is a data manipulation language statement.

- Data Manipulation Language statements or DML statements are used to read and modify data.
- The SELECT statement is called a query, and the output we get from executing this query is called a result set or a result table.
- In its simplest form, a SELECT statement is select star from table name. Based on the book entity example, we would create the table using the entity name book and the entity attributes as the columns of the table.
- The data would be added to the book table by adding rows to the table using the insert statement.

In the book entity example, select star from book gives the result set of four rows. All the data rows for all columns in the table book are displayed.

In addition, you can also retrieve all the rows for all columns by specifying the column names individually in the SELECT statement. You don't always have to retrieve all the columns in a table.

You can retrieve just a subset of columns. If you want, you can retrieve just two columns from the table book. For example book_id and title.

In this case, the select statement is select book_id, title from book. In this case, only the two columns display for each of the four rows. Also notice that the order of the columns displayed

always matches the order in the SELECT statement. However, what if we want to know the title of the book whose book_id is B1.

Relational operation helps us in restricting the result set by allowing us to use the clause WHERE.

- The WHERE clause always requires a predicate.

A predicate

- is conditioned evaluates to true, false or unknown.
- Predicates are used in the search condition of the WHERE clause.

So, if we need to know the title of the book whose book_id is B1, we use the WHERE clause with the predicate book_id equals B1. Select book_id title from book where book_id equals B1.

Notice the result set is now restricted to just one row whose condition evaluates to true.

The previous example used comparison operator equal to in the WHERE clause.

There are other comparison operators supported by a relational database management system: equal to, greater than, less than, greater than or equal to, less than or equal to, and not equal to.

Now you can retrieve data and select columns from a relational database table, define the use of a predicate, identify the syntax of the SELECT statement using the WHERE clause, and

list the comparison operators supported by a relational database management system.

SELECT Statement

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

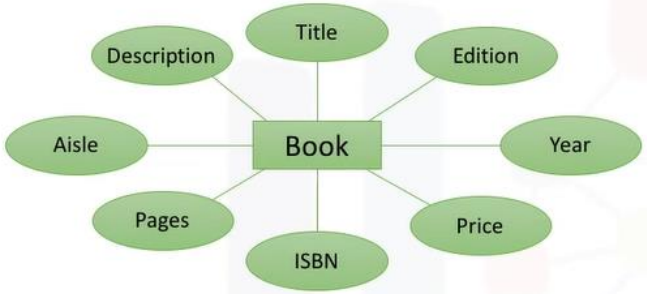
- Retrieve data from a relational database table
- Define the use of a predicate
- Identify the syntax of the SELECT statement using the WHERE clause
- List the comparison operators supported by a RDBMS

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



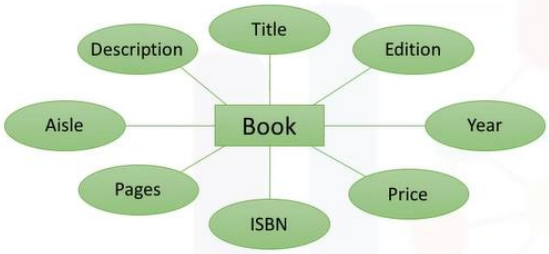
Title	Edition	Year	Price	ISBN	Pages	Aisle	Description
Database Fundamentals	1	2010	24.99	978-0-9800628 3-1-1	300	DB-A02	Teaches you the fundamentals of databases
Getting started with DB2 Express-C	1	2010	24.99	978-0-9866628 3-5-1	280	DB-A01	Teaches you the essentials of DB2 using DB2 Express-C

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
```

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	<>

Summary

Now you can:

- 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

Reading: SELECT statement examples (2 mins)

Objectives

At the end of this lab you will be able to:

- use SELECT queries to retrieve data from the database

Effort: 2 min

Effort: 2 min

The general syntax of SELECT statments is:

select COLUMN1, COLUMN2, ... from TABLE1 ;

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

select * from COUNTRY ;

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

select * from COUNTRY where ID < 5 ;

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

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

In the lab that follows later in the module, you will apply these concepts and practice SELECT queries hands-on.

Good luck!

Hands-on Lab : Basics of SQL SELECT Statement

Estimated time needed: 20 minutes

In this lab, you will learn one of the most commonly used statements of SQL (Structured Query Language), the SELECT statement. The SELECT statement is used to select data from a database.

How does the syntax of a SELECT statement look?

```
SELECT column1, column2, ...
FROM table_name
WHERE condition
;
```

What do the keywords / clauses of a SQL statement shown above do?

- **FROM**: Specifies from which table to get the data. The clause can include optional JOIN subclauses to specify the rules for joining tables.
- [Optional Clause] **WHERE** : Specifies which rows to retrieve.

Why is there a semicolon after the SQL statements?

- Some database systems require a semicolon at the end of each SQL statement for execution. It is a standard way to separate one SQL statement from another which allows more than one SQL statement to be executed in the same call to the server. So, it is good practice to use a semicolon at the end of each SQL statement.

Software Used in this Lab

In this lab, you will use **Datasette** , an open source multi-tool for exploring and publishing data.

Database Used in this Lab

The database used in this lab comes from the following dataset source: **Film Locations in San Francisco** under a **PDDL: Public Domain Dedication and License**.

Objectives

After completing this lab, you will be able to:

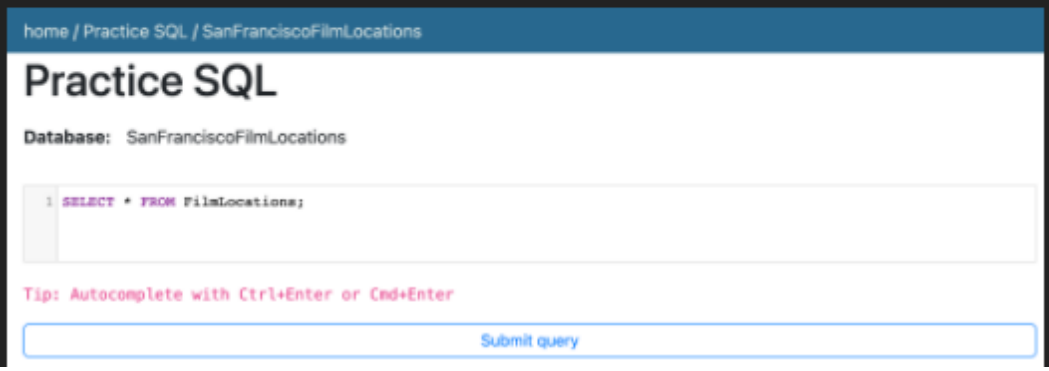
- Query a database
- Retrieve data records from one or more tables of a database as resultset according to the criteria you specify

Task A: Exploring the Database

Let us first explore the **SanFranciscoFilmLocations** database using the **Datasette** tool:

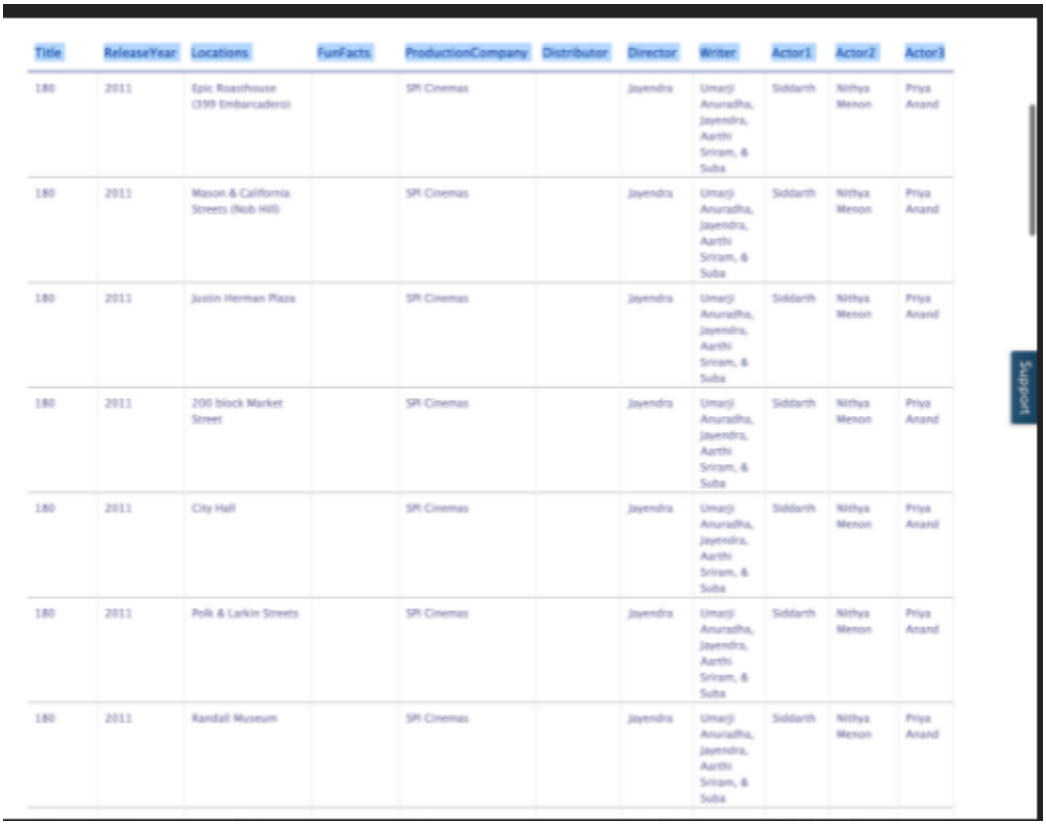
1. If the first statement listed below is not already in the Datasette textbox on the right, then copy the code below by clicking on the little copy button on the bottom right of the codeblock below and then paste it into the textbox of the Datasette tool using either **Ctrl+V** or right-click in the text box and choose **Paste**.

```
SELECT * FROM FilmLocations;
```



2. Click **Submit Query**.

3. Now you can scroll down the table and explore all the columns and rows of the **FilmLocations** table to get an overall idea of the table contents.



Title	ReleaseYear	Locations	FunFacts	ProductionCompany	Distributor	Director	Writer	Actor1	Actor2	Actor3
180	2011	Epic Roasthouse (399 Embarcadero)		SPI Cinemas		Jayendra	Umarji Anuradha, Jayendra, Aarthi Sriram, & Suba	Siddarth	Nithya Menon	Priya Anand
180	2011	Mason & California Streets (Nob Hill)		SPI Cinemas		Jayendra	Umarji Anuradha, Jayendra, Aarthi Sriram, & Suba	Siddarth	Nithya Menon	Priya Anand
180	2011	Justin Herman Plaza		SPI Cinemas		Jayendra	Umarji Anuradha, Jayendra, Aarthi Sriram, & Suba	Siddarth	Nithya Menon	Priya Anand
180	2011	200 block Market Street		SPI Cinemas		Jayendra	Umarji Anuradha, Jayendra, Aarthi Sriram, & Suba	Siddarth	Nithya Menon	Priya Anand
180	2011	City Hall		SPI Cinemas		Jayendra	Umarji Anuradha, Jayendra, Aarthi Sriram, & Suba	Siddarth	Nithya Menon	Priya Anand
180	2011	Polk & Larkin Streets		SPI Cinemas		Jayendra	Umarji Anuradha, Jayendra, Aarthi Sriram, & Suba	Siddarth	Nithya Menon	Priya Anand
180	2011	Randall Museum		SPI Cinemas		Jayendra	Umarji Anuradha, Jayendra, Aarthi Sriram, & Suba	Siddarth	Nithya Menon	Priya Anand

4. These are the column attribute descriptions from the **FilmLocations** table:

```
FilmLocations(
    Title:              titles of the films,
    ReleaseYear:        time of public release of the films,
    Locations:          locations of San Francisco where the films were shot,
    FunFacts:           funny facts about the filming locations,
    ProductionCompany:  companies who produced the films,
    Distributor:        companies who distributed the films,
    Director:           people who directed the films,
    Writer:             people who wrote the films,
    Actor1:             person 1 who acted in the films,
    Actor2:             person 2 who acted in the films,
    Actor3:             person 3 who acted in the films
)
```

Task B: Example exercises on SELECT statement

Now let us go through some examples of SELECT queries:

1. In this example, suppose we want to retrieve details of all the films from the "FilmLocations" table. The details of each film record should contain all the film columns.
    1. Problem:

        > *Retrieve all records with all columns from the "FilmLocations" table.*

home / Practice SQL / SanFranciscoFilmLocations

Practice SQL

Database: SanFranciscoFilmLocations

```
1 SELECT * FROM FilmLocations;
2 SELECT Title, Director, Writer FROM FilmLocations;
3 SELECT Title, ReleaseYear, Locations FROM FilmLocations WHERE ReleaseYear>=2001;
4 SELECT Funfacts, Locations FROM FilmLocations;
5 SELECT Title, ReleaseYear, Locations FROM FilmLocations WHERE ReleaseYear<=2000;
6 SELECT Title, ProductionCompany, Locations, ReleaseYear FROM FilmLocations WHERE Writer <> "Jame
```

Tip: Autocomplete with Ctrl+Enter or Cmd+Enter

Submit query

Practice SQL

Database: SanFranciscoFilmLocations

```
OM FilmLocations;
e, Director, Writer FROM FilmLocations;
e, ReleaseYear, Locations FROM FilmLocations WHERE ReleaseYear>=2001;
acts, Locations FROM FilmLocations;
e, ReleaseYear, Locations FROM FilmLocations WHERE ReleaseYear<=2000;
e, ProductionCompany, Locations, ReleaseYear FROM FilmLocations WHERE Writer <> "James Cameron";
```

Tip: Autocomplete with Ctrl+Enter or Cmd+Enter

Submit query

COUNT, DISTINCT, LIMIT

In this video, we'll briefly present a few useful expressions that are used with select statements.

The first one is COUNT,

- COUNT is a built-in database function that retrieves the number of rows that match the query criteria.

For example, get the total number of rows in a given table, select COUNT(*) from tablename.

Let's say you create a table called MEDALS which has a column called COUNTRY, and

you want to retrieve the number of rows where the medal recipient is from Canada.

You can issue a query like this:

Select COUNT(COUNTRY) from MEDALS where COUNTRY='CANADA.'

The second expression is DISTINCT.

- DISTINCT is used to remove duplicate values from a result set.

Example, to retrieve unique values in a column, select DISTINCT columnname from tablename.

In the MEDALS table mentioned earlier, a country may have received a gold medal multiple times.

Example, retrieve the list of unique countries that received gold medals.

That is, removing all duplicate values of the same country.

Select DISTINCT COUNTRY from MEDALS where MEDALTYPE = 'GOLD'.

The third expression is LIMIT,

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

Example, retrieve just the first 10 rows in a table.

Select * from tablename LIMIT 10.

This can be very useful to examine the results set by looking at just a few rows instead of retrieving the entire result set which may be very large.

Example, retrieve just a few rows in the MEDALS table for a particular year.

Select * from MEDALS where YEAR = 2018 LIMIT 5.

In this video we looked at some useful expressions that are used with select

statements, namely the COUNT, DISTINCT, and LIMIT built-in functions.

COUNT, DISTINCT, LIMIT

Rav Ahuja

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 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 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

n this video, we will learn about populating a relational database table.

At the end of this video, you'll be able to identify the syntax of the INSERT statement and explain two methods to add rows to a table.

After table is created, the table needs to be populated with data.

To insert data into a table, we use the INSERT statement.

- The INSERT statement is used to add new rows to a table.
- The INSERT statement is one of the data manipulation language statements.

Data manipulation language statements or DML statements are used to read and modify data.

Based on the author entity example, we created the table using the entity name, author, and the entity attributes as the columns of the table.

Now we will add the data to the author table by adding rows to the table.

To add the data to the author table, we use the INSERT statement.

The syntax of the INSERT statement looks like this,

INSERT INTO [TableName] <([ColumnName],...)> VALUES ([Value],...)

In this statement, table name identifies the table, the column name list identifies each column in the table, and the values clause specifies the data values to be added to the columns in the table.

To add a row with the data for Raul Chong, we insert a row with an author underscore ID of A one, the last name is Chong,

the first name as Raul, the email as RFC@IBM.com,

the city as Toronto, and the country as CA for Canada.

The author table has six columns, so the INSERT statement lists the six column

names separated by commas, followed by a value for each of the columns also separated by commas.

It is important that the number of values provided in the values clause

is equal to the number of column names specified in the column name list.

This ensures that each column has a value.

Tables do not need to be populated one row at a time.

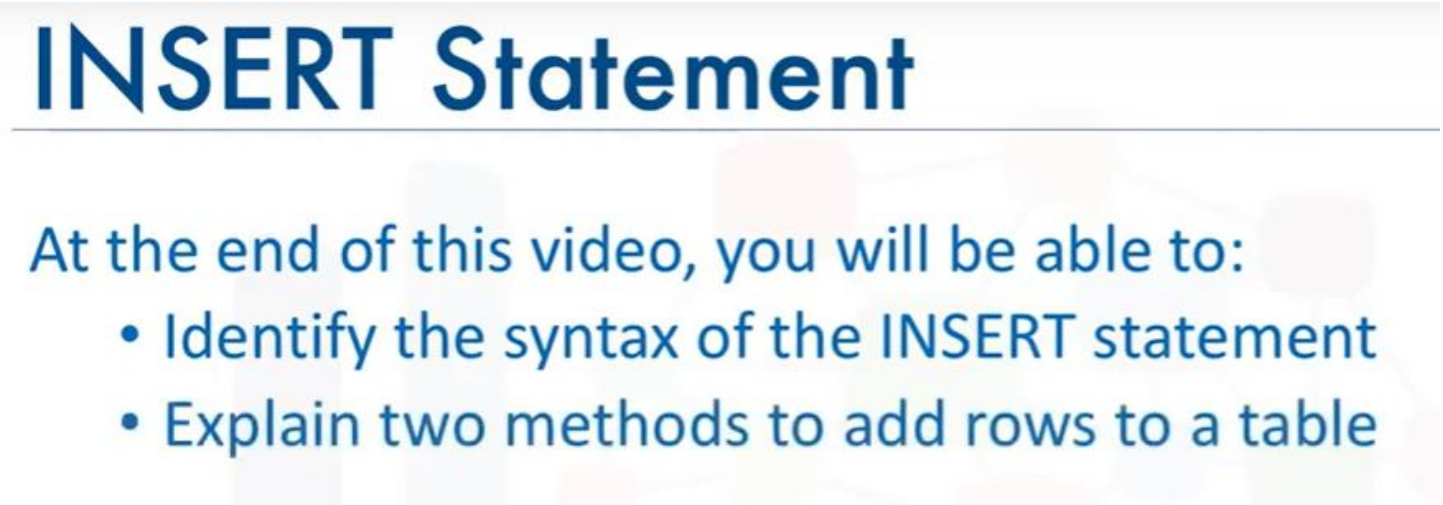
Multiple rows can be inserted by specifying each row in the values clause.

In the values clause, each row is separated by a comma.

For example, in this INSERT statement we are inserting two rows, one for Raul Chong and one for Rav Ahuja.

Now you can identify the syntax of the INSERT statement, and explain the two methods to add rows to a table.

One row at a time or multiple rows.

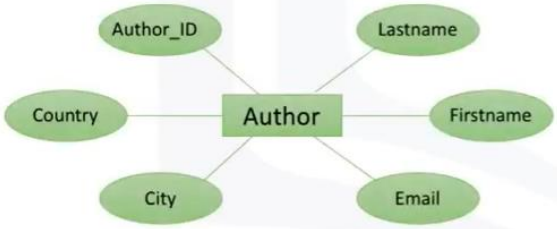


Adding rows to a table

- Create the table (CREATE TABLE statement)
- Populate table with data:
  - INSERT statement

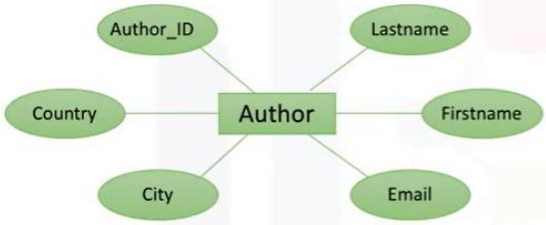
Adding rows to a table

- 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

Using the INSERT Statement

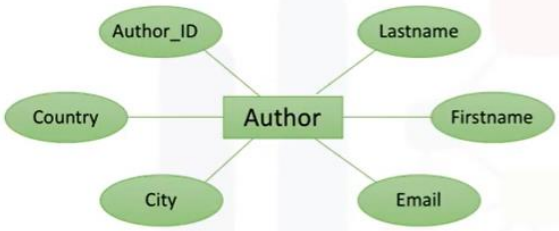


Author_ID	Lastname	Firstname	Email	City	Country
A1	Chong	Raul	rfc@ibm.com	Toronto	CA
A2	Ahuja	Rav	ra@ibm.com	Toronto	CA
A3	Hakes	Ian	ih@ibm.com	Toronto	Ca
A4	Sharma	Neeraj	ns@ibm.com	Chennai	IN
A5	Perniu	Liviu	lp@ibm.com	Transylvania	RO

```
INSERT INTO [TableName]
    <([ColumnName],...)>
VALUES ([Value],...)
```

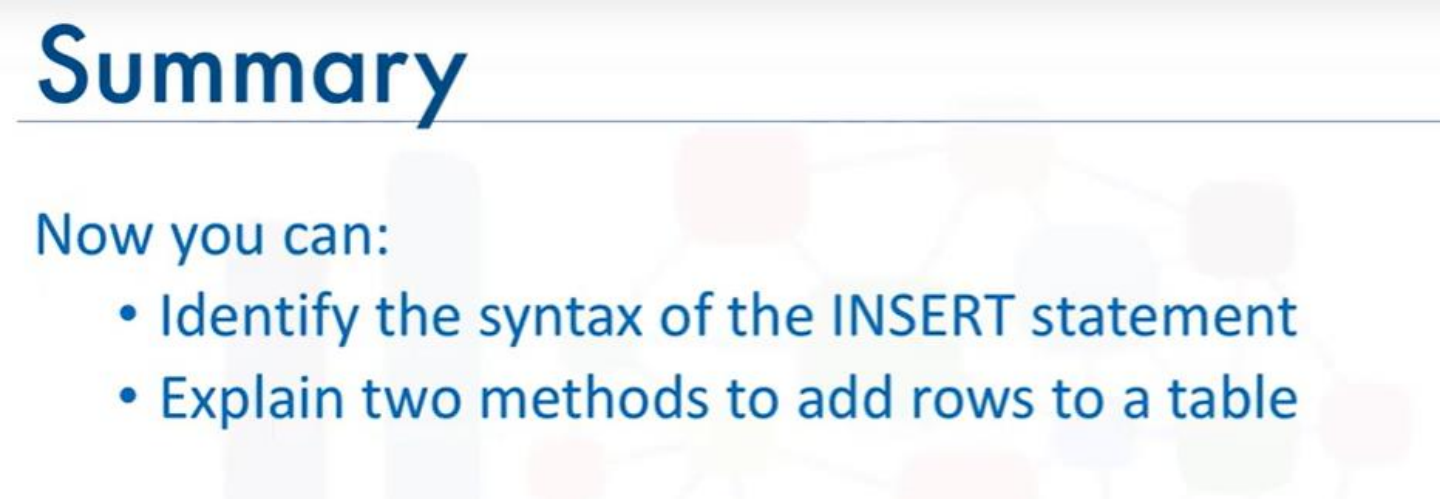
```
INSERT INTO AUTHOR
    (AUTHOR_ID, LASTNAME, FIRSTNAME, EMAIL, CITY, COUNTRY)
VALUES ('A1', 'Chong', 'Raul', 'rfc@ibm.com', 'Toronto', 'CA')
```

Inserting multiple rows



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

```
INSERT INTO AUTHOR
    (AUTHOR_ID, LASTNAME, FIRSTNAME, EMAIL, CITY, COUNTRY)
VALUES
    ('A1', 'Chong', 'Raul', 'rfc@ibm.com', 'Toronto', 'CA')
    ('A2', 'Ahuja', 'Rav', 'ra@ibm.com', 'Toronto', 'CA')
```



UPDATE and DELETE Statements

Hello and welcome to the UPDATE Statement and the DELETE Statement.

In this video, we will learn about altering and deleting data in a relational database table.

At the end of this lesson, you will be able to identify the syntax of the UPDATE statement and DELETE statement, and explain the importance of the WHERE clause in these statements.

After a table is created and populated with data, the data in a table can be altered with the UPDATE statement.

- The UPDATE statement is one of the data manipulation language or DML statements.
- DML statements are used to read and modify data.
- Based on the author entity example, we created the table using the entity name author and the entity attributes as the columns of the table.
- Rows were added to the author table to populate the table.

Sometime later, you want to alter the data in the table.

To alter or modify the data in the author table, we use the UPDATE statement.

The syntax of the UPDATE statement looks like this: UPDATE [TableName] SET [[ColumnName]=[Value]] In this statement, table name identifies the table, the column name identifies the column value to be changed as specified in the WHERE condition.

Let's look at an example.

In this example, you want to update the first name and last name of the author with Author_Id A2 from Rav Ahuja to Lakshmi Katta.

In this example, to see the update statement in action, we start by selecting all rows from the author table to see the values.

To change the first name and last name to Lakshmi Katta, where the author ID is equal

to A2, enter the UPDATE statement as follows: UPDATE AUTHOR SET LASTNAME='KATTA' FIRSTNAME='LAKSHMI'

WHERE AUTHOR_ID='A2' Now, to see the result of the update, select

all rows again from the author table and you will see that in row two the name changed

from Rav Ahuja to Lakshmi Katta.

Note that if you do not specify the WHERE clause, all the rows in the table will be

updated.

In this example, without specifying the WHERE clause, all rows in the table would have changed

the first and last names to Lakshmi Kata.

Sometime later, there might be a need to remove one or more rows from a table.

The rows are removed with the DELETE statement.

- The DELETE statement is one of the data manipulation language statements used to read and modify data.
- The syntax of the DELETE statement looks like this: DELETE FROM [TableName] WHERE [Condition]
- The rows to be removed are specified in the WHERE condition.

Based on the author entity example, we want to delete the rows for author ID A2 and A3.

Let's look at an example.

DELETE FROM AUTHOR WHERE AUTHOR_ID ID IN ('A2', 'A3') Note that if you do not

specify the WHERE clause, all the rows in the table will be removed.

Now, you can identify the syntax of the UPDATE statement and DELETE statement, and explain

the importance of the WHERE clause in these statements.

UPDATE and 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

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

Deleting Rows from a table

- Remove 1 or more rows from the table:
  - DELETE statement
    - A DML statement used to read and modify data

DELETE FROM [TableName]
<WHERE [Condition]>

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

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

Summary

Now you can:

- 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

Summary & Highlights

Bookmark this page

Congratulations! You have completed this lesson. At this point in the course, you know:

- You can use Data Manipulation Language (DML) statements to read and modify data.
- The search condition of the WHERE clause uses a predicate to refine the search.
- COUNT, DISTINCT, and LIMIT are expressions that are used with SELECT statements.
- INSERT, UPDATE, and DELETE are DML statements for populating and changing tables.

Module Introduction & Learning Objectives

Bookmarked

Module Introduction

In this module, you will explore the fundamental concepts behind databases, tables, and the relationships between them. You will then create an instance of a database, discover SQL statements that allow you to create and manipulate tables, and then practice them on your own live database.

Learning Objectives

- Describe basic relational database concepts including tables, primary keys, and foreign keys.
- Create a database instance on the Cloud
- Distinguish between Data Definition Language and Data Manipulation Language.
- Explain the syntax of the CREATE TABLE statement.
- Explain the syntax of the ALTER, DROP, and TRUNCATE statements.
- Compose and execute CREATE TABLE, ALTER, DROP, and TRUNCATE statements hands-on on a live database.

Module 2 - Introduction to Relational Databases and Tables

Relational Database Concepts

Hello and welcome to Database Concepts.

In this video, we will learn about different types of models, how we use models to map data to tables, and define relationships between tables.

At the end of this lesson, you will be able to explain the advantage of the relational model, explain how the entity name and attributes map to a relational database table, describe the difference between an entity and an attribute, identify some commonly used data types, and describe the function of primary keys.

The relational model is the most used data model for databases because this model allows for data independence.

Data is stored in a simple data structure.

Tables: this provides logical data independence, physical data independence, and physical storage independence.

An entity relationship data model, or ER data model, is an alternative to a relational data model.

Using a simplified library database as an example, this figure shows an entity relationship diagram or ERD that represents entities called tables and their relationships.

In the library example, we have books. A book can be written by one or many authors.

The library can have one or many copies of a book.

Each copy can be borrowed by only one borrower at a time.

An entity relationship model proposes thinking of a database as a collection of entities rather than being used as a model on its own.

The ER model is used as a tool to design relational databases.

In the ER model, entities are objects that exist independently of any other entities in the database.

The building blocks of an ER diagram are entities and attributes.

An entity

- can be a noun: person, place, or thing.
- In an ER diagram, an entity is drawn as a rectangle.

- Entities have attributes which are the data elements that characterize the entity.

Attributes

- tell us more about the entity.

In an ER diagram, attributes are drawn as ovals.

Using a simplified library as an example, the book is an example of an entity.

- Attributes are certain properties or characteristics of an entity and tell us more about the entity.

The entity book has attributes such as book title, the edition of the book, the year the book was written, etc.

- Attributes are connected to exactly one entity.

The entity book becomes a table in the database and the attributes become the columns in a table.

- A table is a combination of rows and columns.

While mapping, the entity becomes the table.

Having said that, the table has not yet taken the form of rows and columns.

The attributes get translated into columns in a table providing the actual table form of rows and columns.

Later, we add some data values to each of the columns, which completes the table form.

Each attribute stores data values of different formats, characters, numbers dates, currency, and many more besides.

In the book table example, the title is made up of characters.

As book titles vary in length, we can set the variable character data type for the title column: Varchar.

For character columns that do not vary in length, we use character or Char.

The Edition and year columns would be numeric.

The ISBN column would be Char because it contains dashes as well as numbers and so on.

Using the book entity mapping as an example, we can create the tables for the remainder of our simplified library example using entity names, like author, author list, borrower, loan, and copy.

The entity attributes will be the columns of the tables.

Each table is assigned a primary key.

The primary key of a relational table uniquely identifies each tuple or row in a table, preventing duplication of data and providing a way of defining relationships between tables.

Tables can also contain foreign keys which are primary keys defined in other tables, creating a link between the tables.

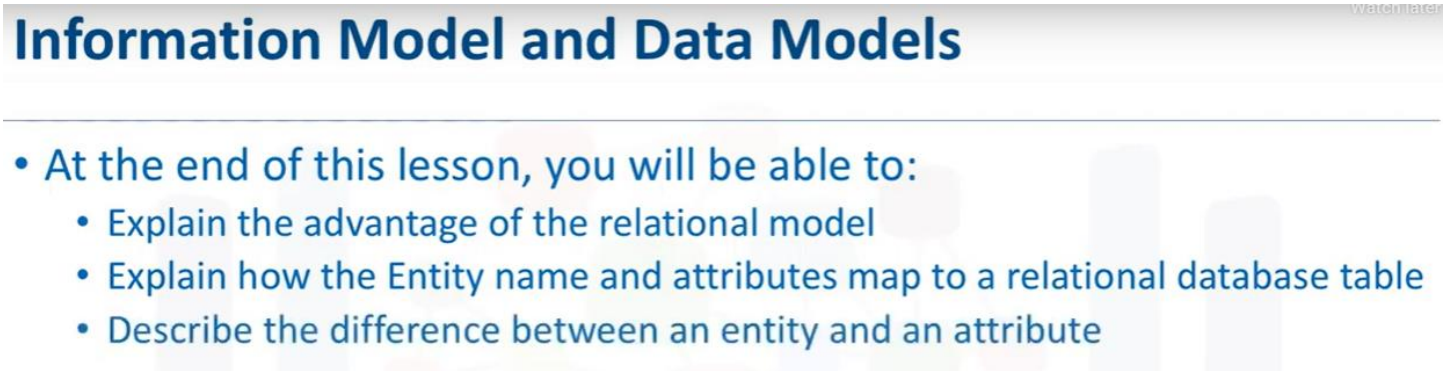
Now you know that the key advantage of the relational model is logical and physical data independence and storage independence.

Entities are independent objects which can have multiple characteristics called attributes.

When mapping to a relational database, entities are represented as tables and attributes map to columns.

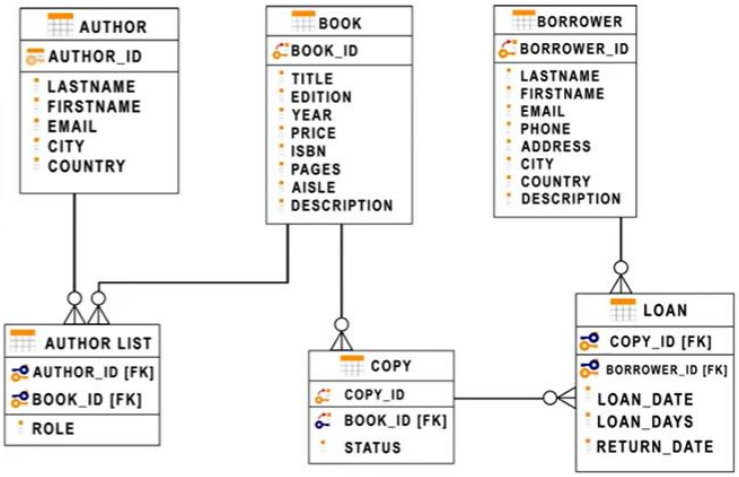
Common data types include characters such as Char and Varchar, numbers such as integer and decimal, and timestamps including date and time.

A primary key uniquely identifies a specific row in a table and prevents duplication of data.



Relational Model

- Most used data model
- Allows for data independence
- Data is stored in a tables

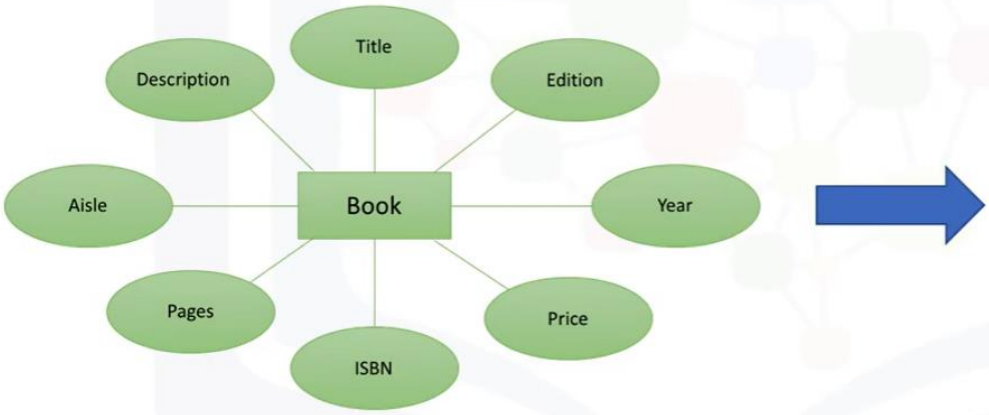


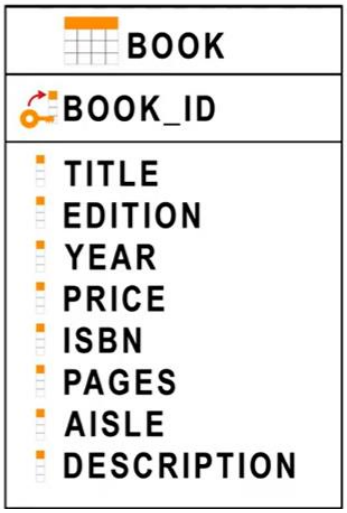
logical data independence - physical data independence - physical storage independence

DS 5 T 2 TA Relational Database Concepts IV2

Entity-Relationship Model

- Used as a tool to design relational databases





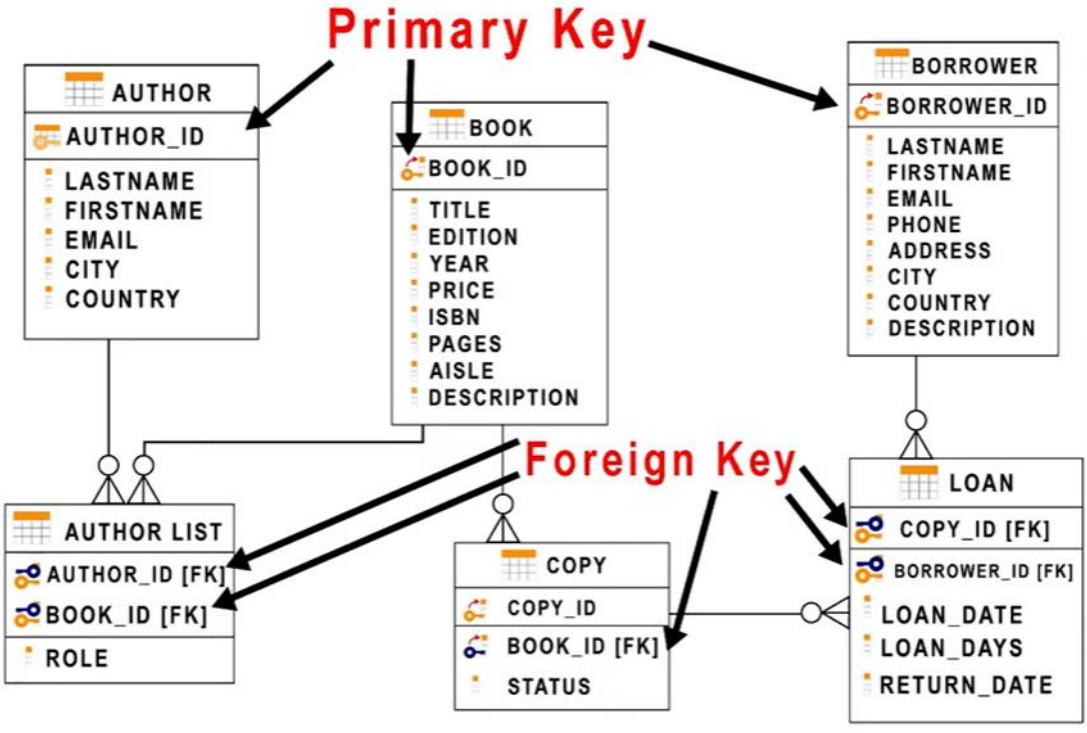
Mapping Entity Diagrams to Tables

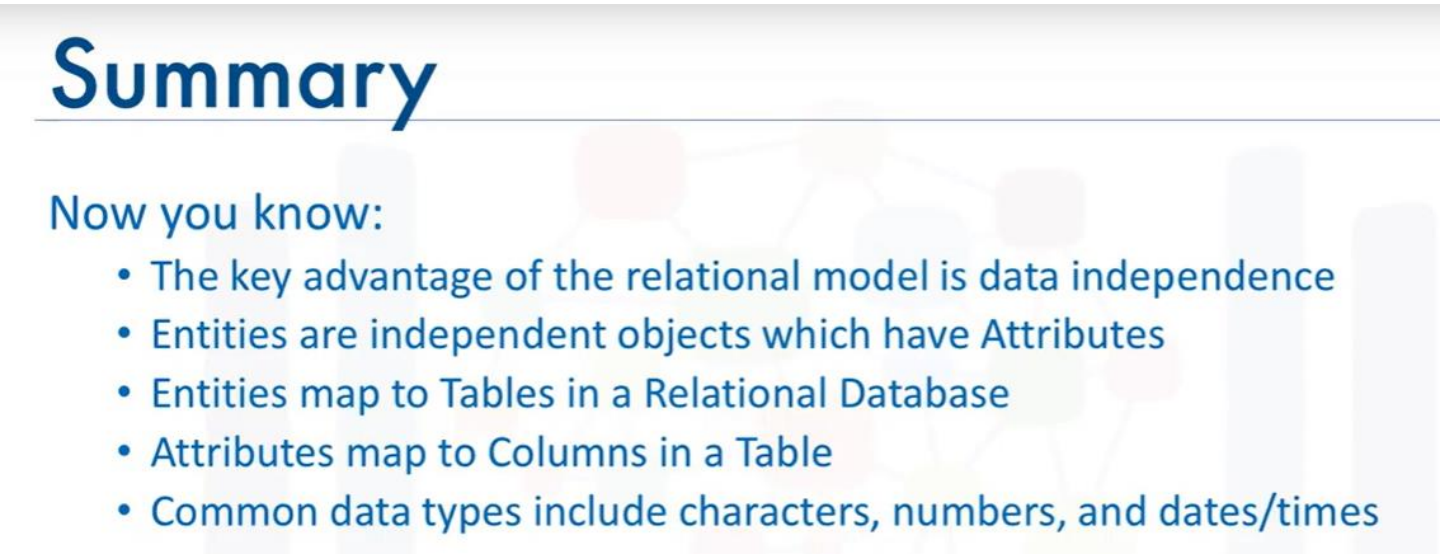
- Entities become tables
- Attributes get translated into columns

Table: Book

Title	Edition	Year	Price	ISBN	Pages	Aisle	Description
Database Fundamentals	1	2010	24.99	978-0-98662 83-1-1	300	DB-A02	Teaches you the fundamentals of databases
Getting started with DB2 Express-C	1	2010	24.99	978-0-9866 283-5-1	280	DB-A01	Teaches you the essentials of DB2 using DB2 Express-C, the free version of DB2

Primary Keys and Foreign Keys





How to Create a Database Instance on Cloud

This video will cover the key concepts around databases in the Cloud.

In order to learn SQL, you first need to have a database available to practice your SQL queries.

An easy way to do so is to create an instance of a database in the Cloud and use it to execute your SQL queries.

After completing this lesson, you will be able to understand basic concepts related to Cloud databases, list a few Cloud databases, describe database service instances, as well as demonstrate how to create a service instance on an IBM Db2 on Cloud.

A Cloud database

- is a database service built and accessed through a Cloud platform.
- It serves many of the same functions as traditional databases with the added flexibility of Cloud computing.

Some advantages of using Cloud databases include;

- ease of use,
- users can access Cloud databases from virtually anywhere using a vendor's API or web interface, or your own applications whether on Cloud or Remote.

Scalability.

- Cloud databases can expand and shrink their storage and compute capacities during runtime to accommodate changing needs and usage demands, so organizations only pay for what they actually use.

Disaster recovery.

- In the event of a natural disaster or equipment failure or power outage, data is kept secure through backups on Remote Servers on Cloud in geographically distributed regions.

- A few examples of relational databases on Cloud include, IBM Db2 on Cloud, databases for PostgreSQL on IBM Cloud, Oracle Database Cloud Service, Microsoft Azure SQL Database, and Amazon Relational Database Services.

- These Cloud databases can run in the Cloud either as a Virtual Machine, which you can manage, or delivered as a Managed Service depending on the vendor. The database services can either be single or multi-tenant depending on the service plan.

To run a database in Cloud, you must first provision an instance of the database service on the Cloud platform of your choice.

An instance of a Database-as-a-Service or DBaaS

- provides users with access to database resources in Cloud without the need for setting up of the underlying hardware, installing the database software, and administering the database.

- The database service instance will hold your data in related tables. Once your data is loaded into the database instance, you can connect to the database instance using a web interface or APIs in your applications.

- Once connected, your application can send SQL queries across to the database instance.
- The database instance then resolves the SQL statements into operations against the data and objects in the database.

Any data retrieved is returned to the application as a result set.

Now let's see how a database instance is created for Db2 on Cloud.

IBM Db2 on Cloud is a

- SQL database provisioned for you in the Cloud. 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. 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.

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".

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

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.

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.

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.

Cloud databases

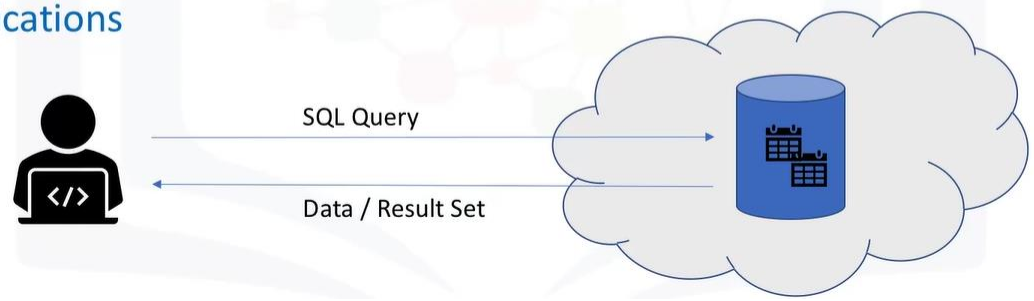
- ✓ Ease of Use and Access
  - API
  - Web Interface
  - Cloud or Remote Applications
- ✓ Scalability & Economics
  - Expand/Shrink Storage & Compute Resources
  - Pay per use
- ✓ Disaster Recovery

Examples of Cloud databases

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

Database service instances

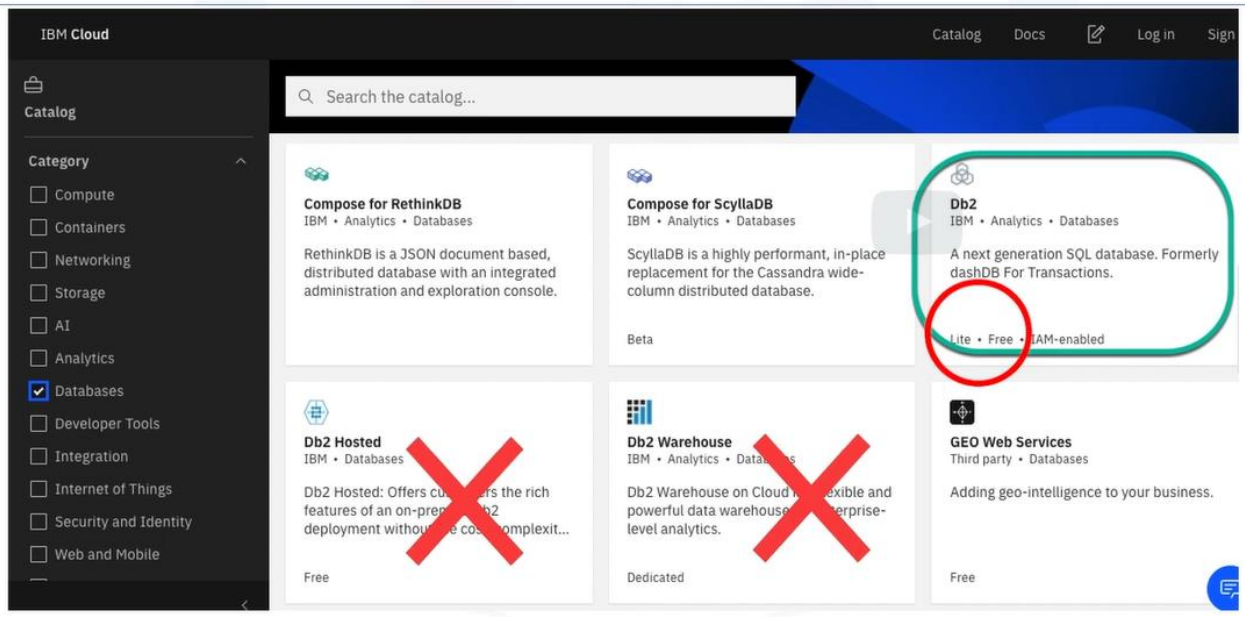
- DBaaS provides users with access to Database resources in cloud without setting up hardware and installing software.
- Database service instance holds data in data objects / tables
- Once data is loaded, it can be queried using web interfaces and applications



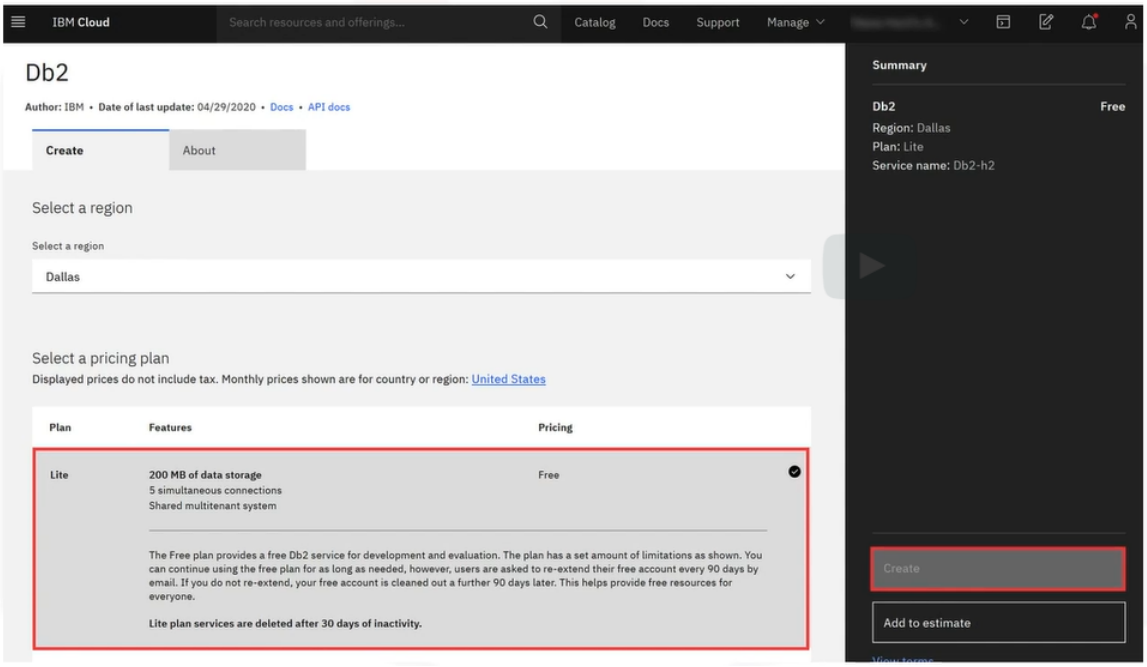
Creating a database instance on IBM Db2 on Cloud

IBM Db2 on Cloud

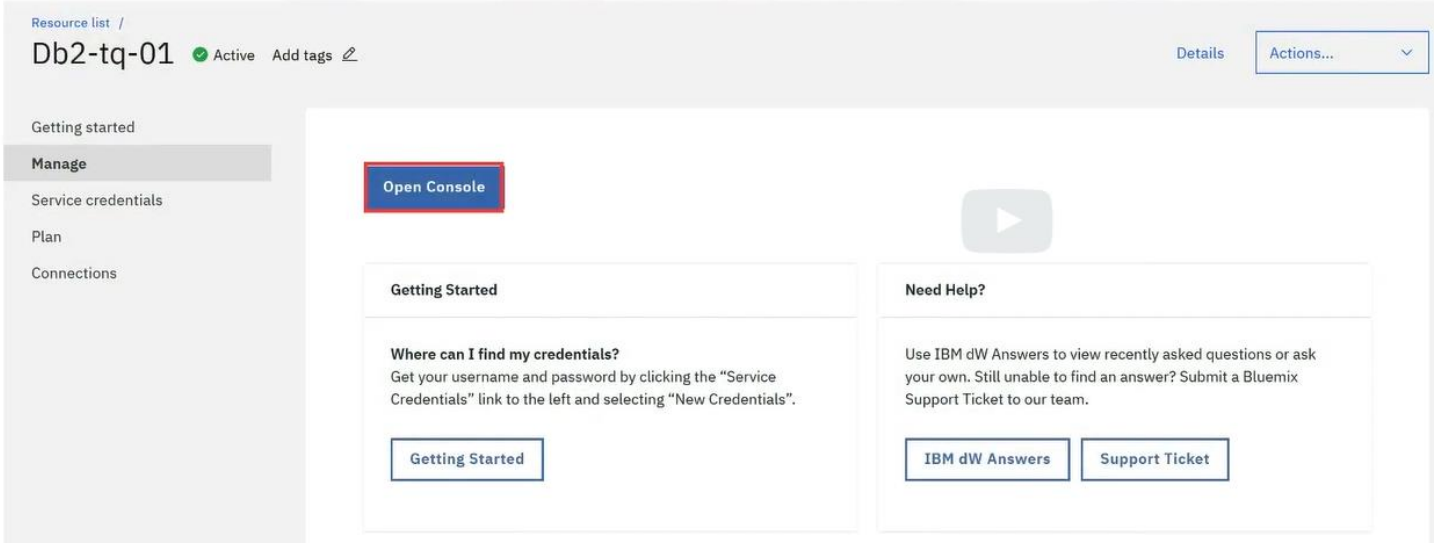
Deploy an instance of Db2 on Cloud Service



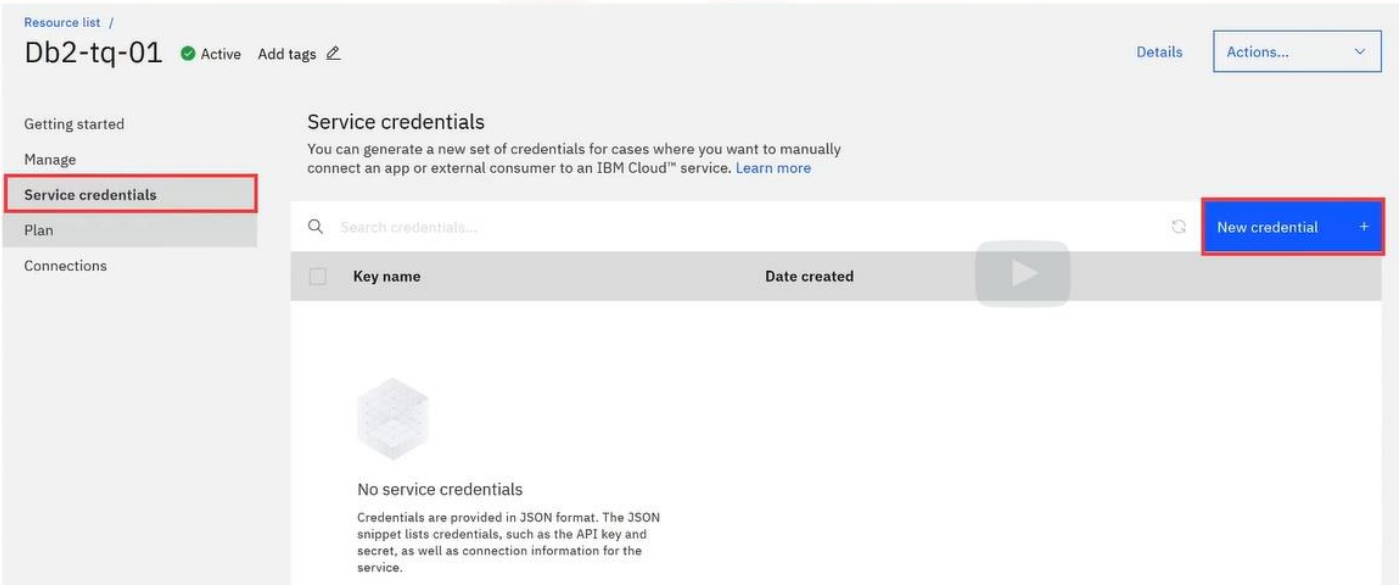
Create a new service



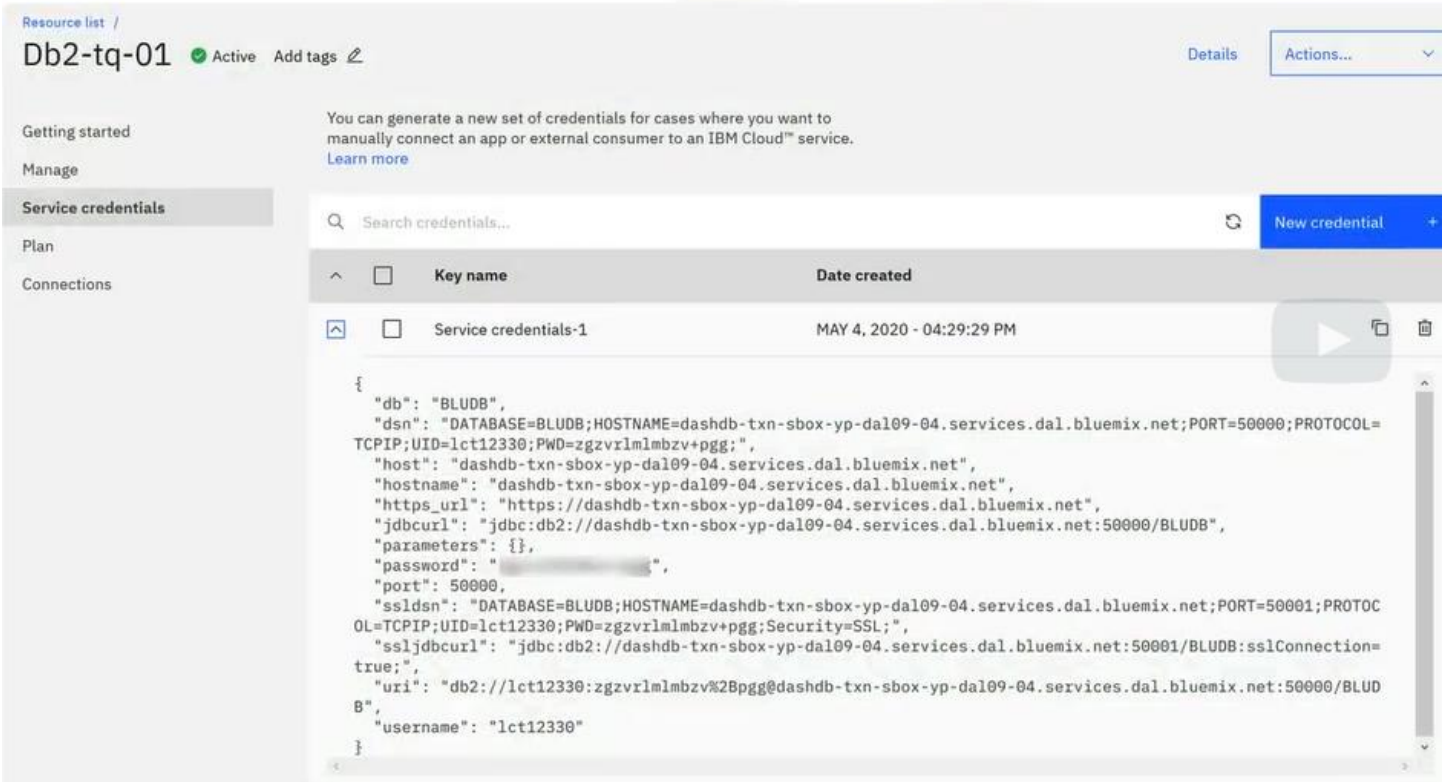
Manage the database instance



Create new service credentials

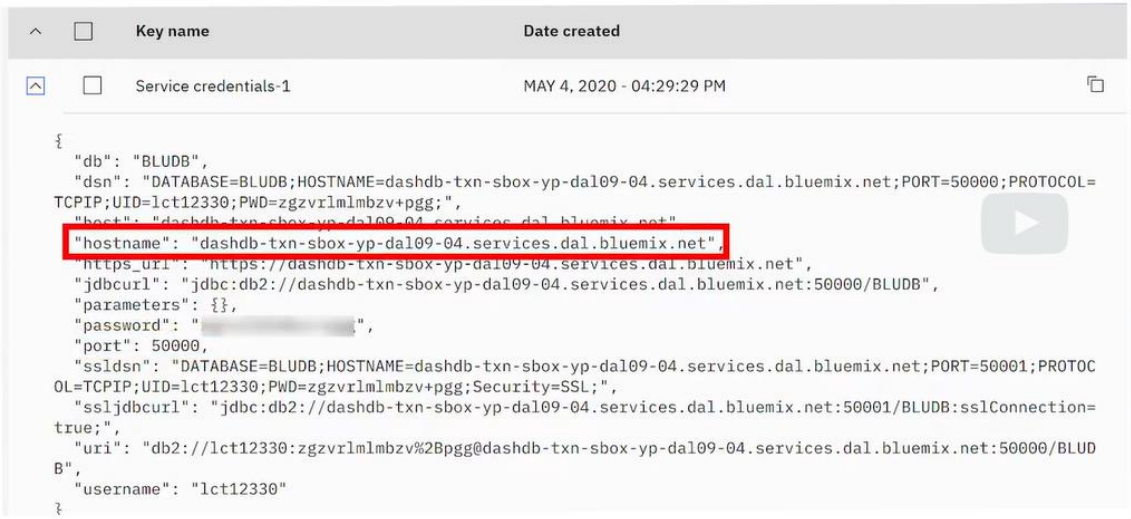


Service credentials



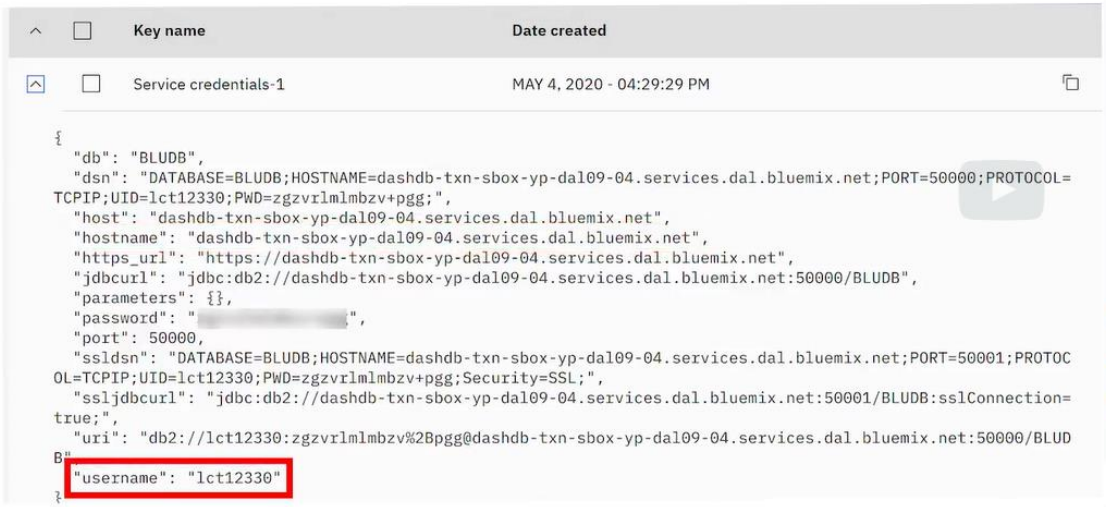
Service credentials

Service credentials



- **Database:** BLUDB
- **Port:** 50000
- **Hostname:** dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net

Service credentials

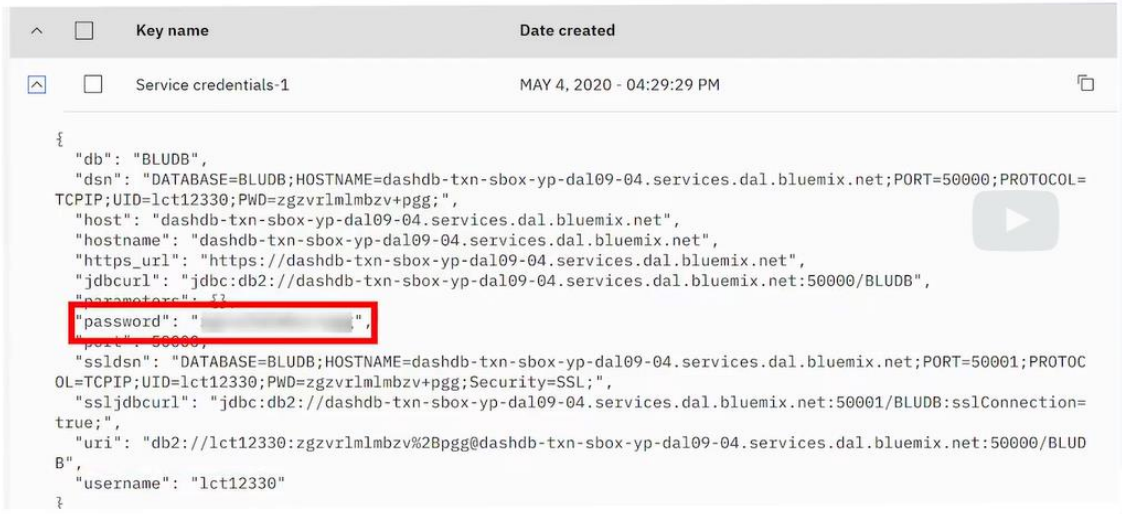


Key name	Date created
Service credentials-1	MAY 4, 2020 - 04:29:29 PM


```
{
  "db": "BLUDB",
  "dsn": "DATABASE=BLUDB;HOSTNAME=dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net;PORT=50000;PROTOCOL=
TCPIP;UID=lct12330;PWD=zgzvrlmlmbzv+pgg;",
  "host": "dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net",
  "hostname": "dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net",
  "https_url": "https://dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net",
  "jdbcurl": "jdbc:db2://dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50000/BLUDB",
  "parameters": {},
  "password": "[redacted]",
  "port": 50000,
  "ssldsn": "DATABASE=BLUDB;HOSTNAME=dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net;PORT=50001;PROTOC
OL=TCPIP;UID=lct12330;PWD=zgzvrlmlmbzv+pgg;Security=SSL;",
  "ssljdbcurl": "jdbc:db2://dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50001/BLUDB:sslConnection=
true;",
  "uri": "db2://lct12330:zgzvrlmlmbzv%2Bpgg@dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50000/BLUD
B",
  "username": "lct12330"
}
```


- **Database:** BLUDB
- **Port:** 50000
- **Hostname:** dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net
- **Username:** lct12330

Service credentials



Key name	Date created
Service credentials-1	MAY 4, 2020 - 04:29:29 PM


```
{
  "db": "BLUDB",
  "dsn": "DATABASE=BLUDB;HOSTNAME=dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net;PORT=50000;PROTOCOL=
TCPIP;UID=lct12330;PWD=zgzvrlmlmbzv+pgg;",
  "host": "dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net",
  "hostname": "dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net",
  "https_url": "https://dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net",
  "jdbcurl": "jdbc:db2://dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50000/BLUDB",
  "parameters": {},
  "password": "[redacted]",
  "port": 50000,
  "ssldsn": "DATABASE=BLUDB;HOSTNAME=dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net;PORT=50001;PROTOC
OL=TCPIP;UID=lct12330;PWD=zgzvrlmlmbzv+pgg;Security=SSL;",
  "ssljdbcurl": "jdbc:db2://dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50001/BLUDB:sslConnection=
true;",
  "uri": "db2://lct12330:zgzvrlmlmbzv%2Bpgg@dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50000/BLUD
B",
  "username": "lct12330"
}
```


- **Database:** BLUDB
- **Port:** 50000
- **Hostname:** dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net
- **Username:** lct12330
- **Password:** *******

IBM Developer
SKILLS NETWORK

Hands-on Lab: Create Db2 service instance and Get started with the Db2 console

Estimated time needed: 15 minutes

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 dc 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.

Objectives

After completing this lab, you will be able to:

- Use IBM cloud account to create and use resources
- Create an instance of a Db2 service
- Locate and explore the Db2 console

Pre-requisites

You will need an IBM Cloud account to do this lab. If you have not created one already, click on this link and follow the instructions to create an IBM Cloud account.

Task 1: Create an instance of IBM Db2 Lite plan

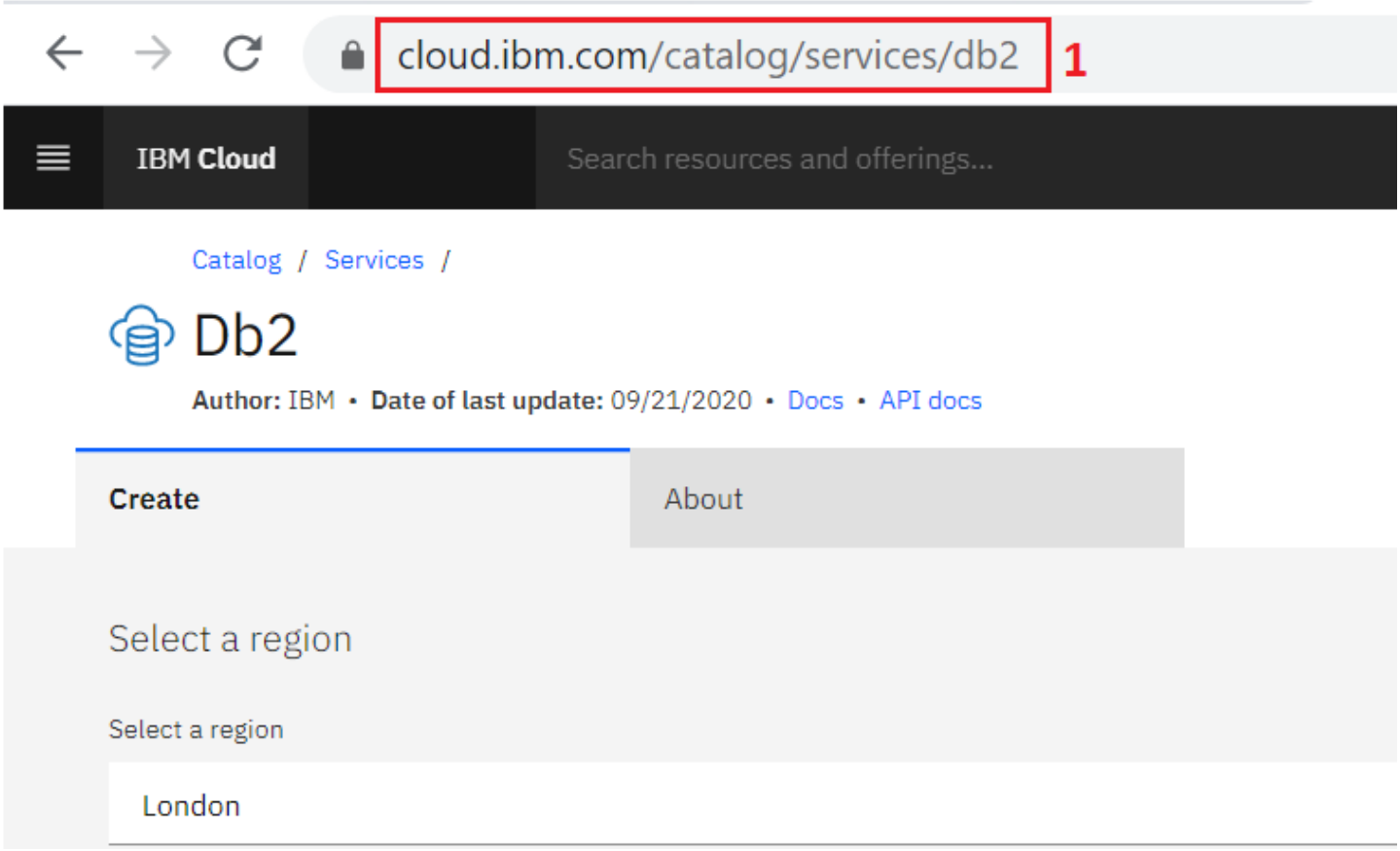
Now let us introduce you to Db2 on IBM Cloud. IBM Db2 is a next generation SQL database provisioned for you in the cloud. You can use Db2 on IBM Cloud just as you would use any database software (RDBMS), but without the overhead and expense of hardware setup or software installation and maintenance. 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, analyze 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.

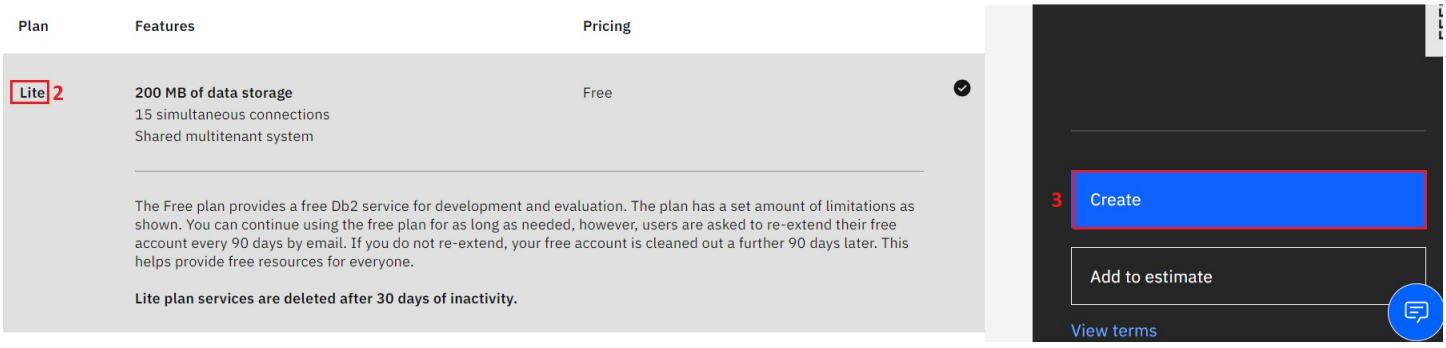
1. Login to IBM Cloud
2. Go to the DB2 Services page on IBM Catalog.
3. Select a location where you want the service to be hosted.

> **Note**: Depending on the Country of your IBM Cloud account a location to deploy will be pre-selected. For example, if you are in the US, the default region 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. Make sure a **Region** is selected as location, not a **Data center.**



4. Scroll down to the Pricing Plans section and select the **Lite plan** (it's a free plan, and available only in **DALLAS** at this point of time) or any other plans as required.

5. 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.

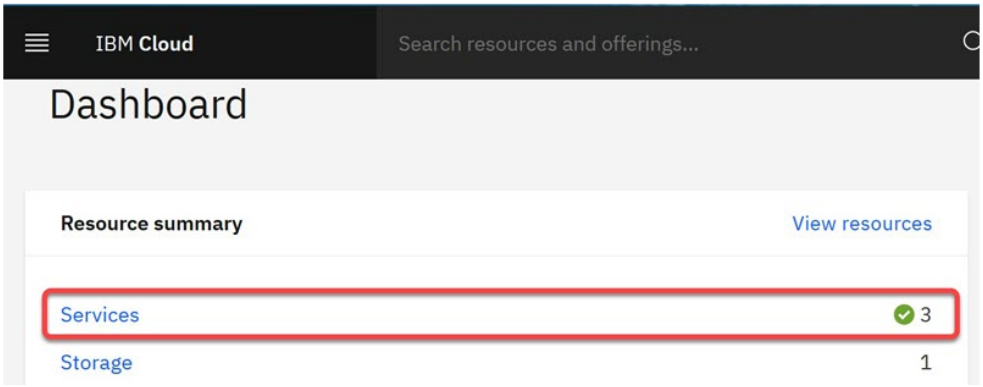


Task 2: Locate and Explore the Db2 console

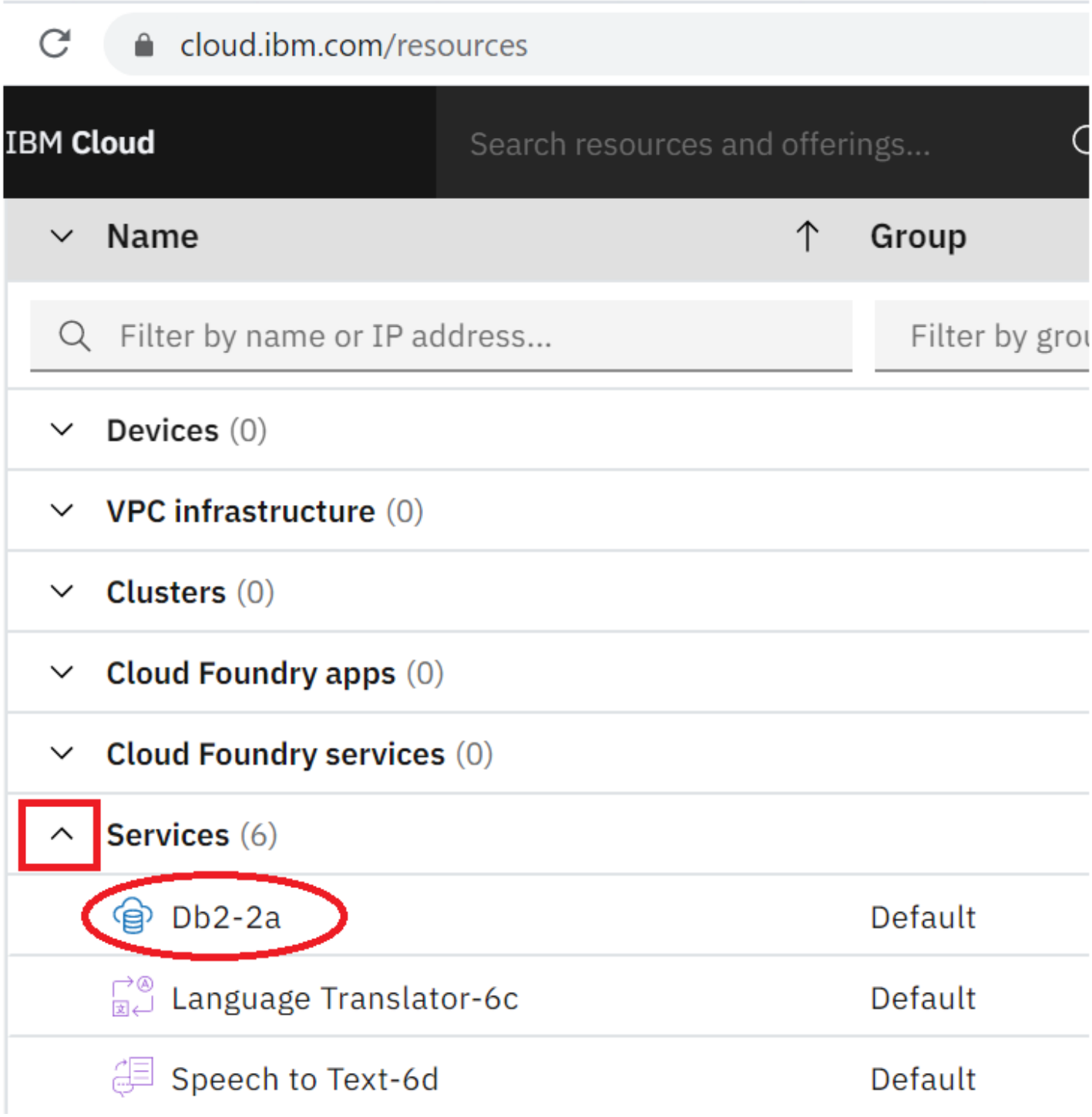
Now that you have created your database instance, you need to know how to get to it, explore the console and start working with it.

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

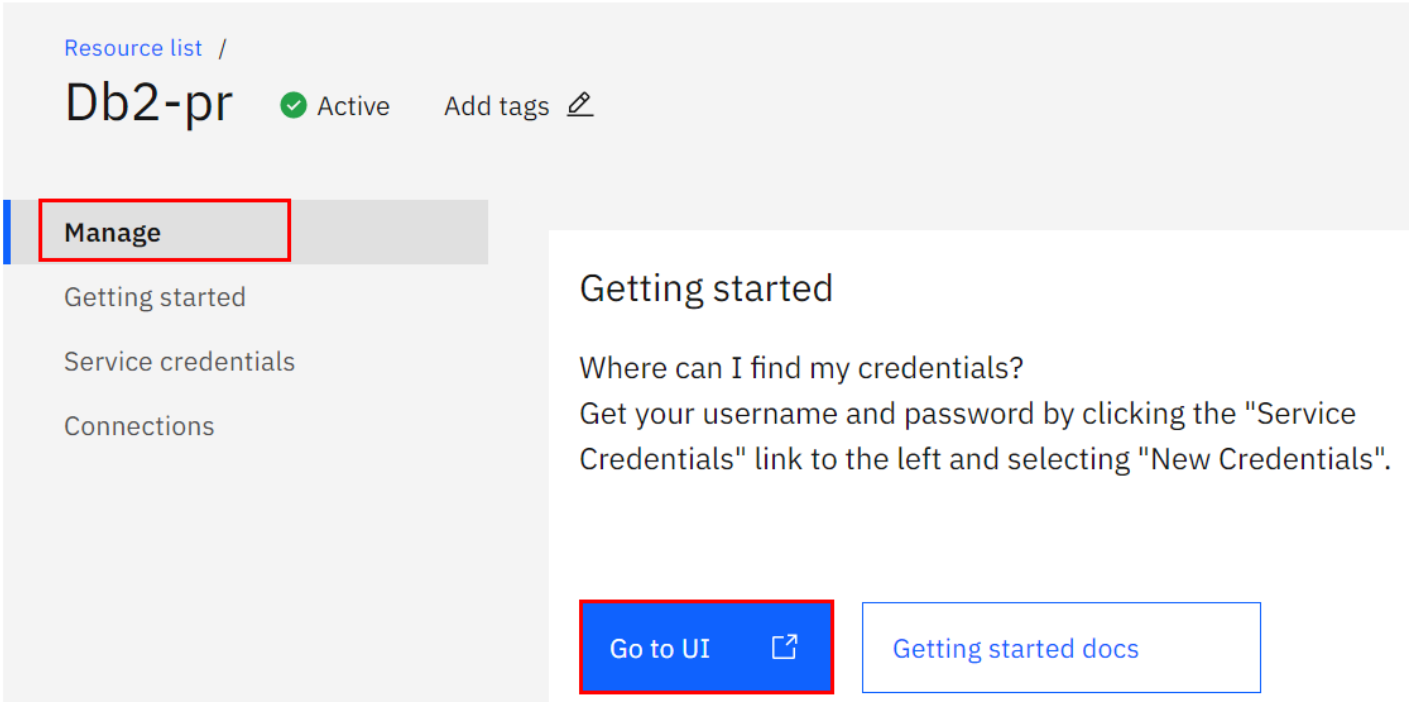
1. To access your database instance, go to your IBM Cloud Resource List (you may need to log into IBM Cloud in the process) directly at: cloud.ibm.com/resources
    - **Alternative**: Go to your IBM Cloud dashboard (you may need to login to IBM Cloud in the process) at: cloud.ibm.com and click **Services**.



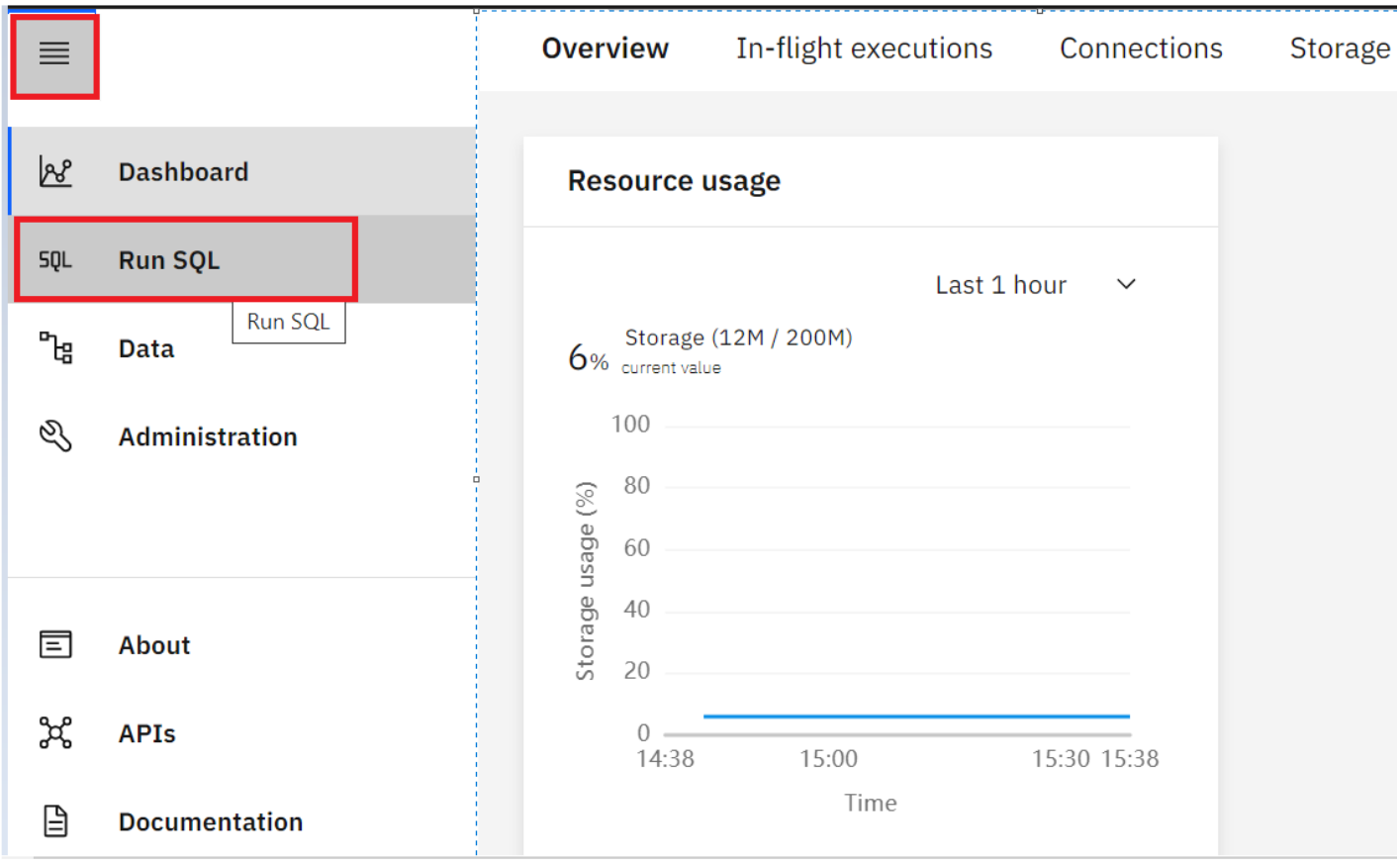
2. In the Resource list, expand the **Services** and locate and click on your instance of Db2 you provisioned in exercise 2 (the name typically starts with Db2-xx for example Db2-fk, Db2-50, etc.)


cloud.ibm.com/resources
IBM Cloud
Search resources and offerings...
Name
Group
Filter by name or IP address...
Filter by gro
Devices (0)
VPC infrastructure (0)
Clusters (0)
Cloud Foundry apps (0)
Cloud Foundry services (0)
Services (6)
Db2-2a
Default
Language Translator-6c
Default
Speech to Text-6d
Default

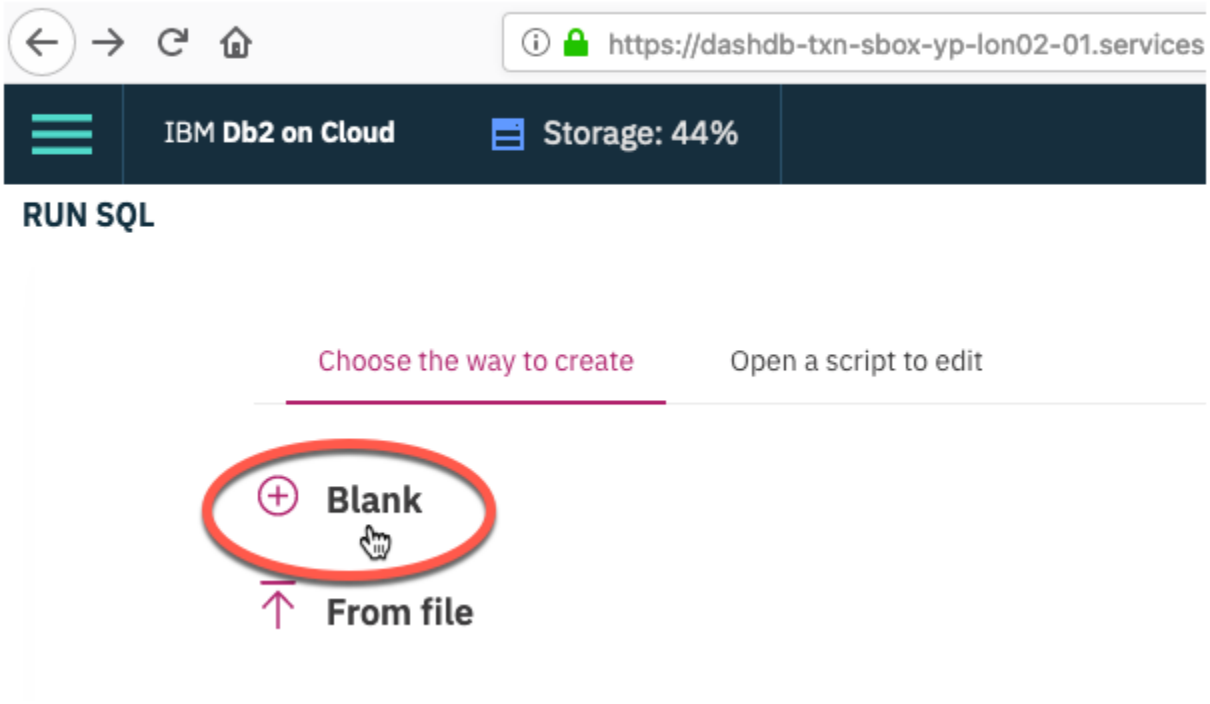
3. Click on the **Go to UI** button.



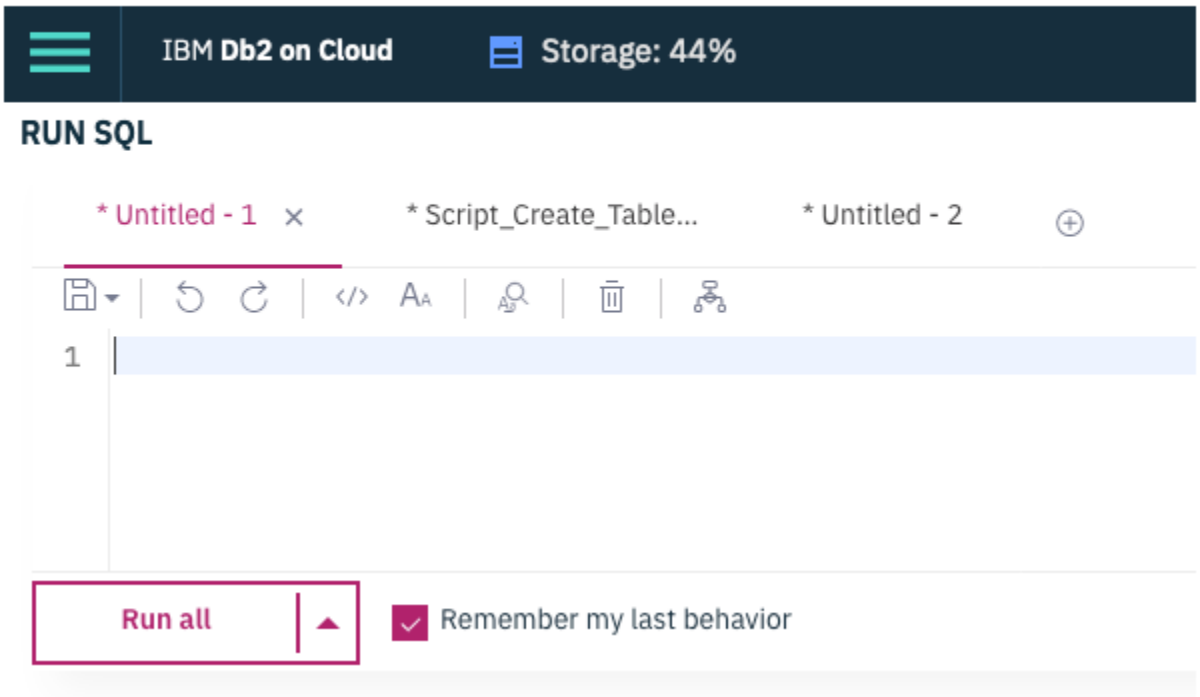
4. The Db2 console will open in a new tab in your web browser. Click on the 3-bar menu icon in the top left corner and then click on **RUN SQL**.



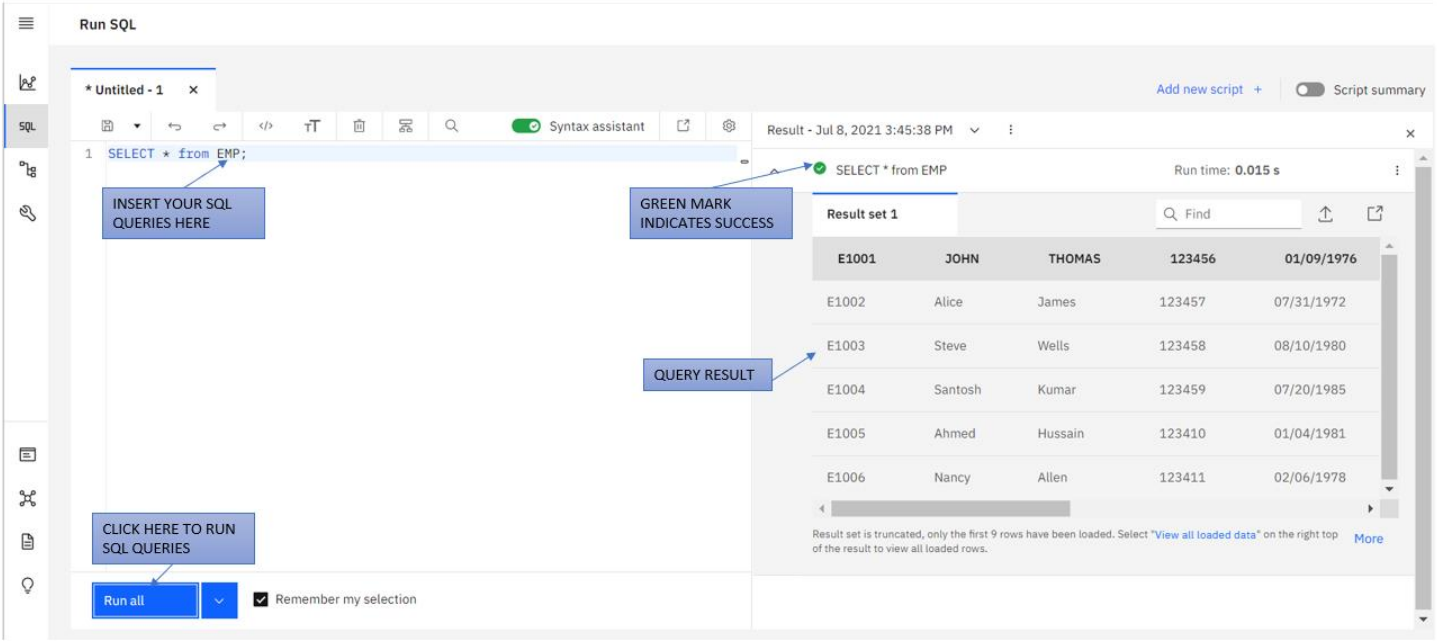
5. On the next screen click on **Blank**.



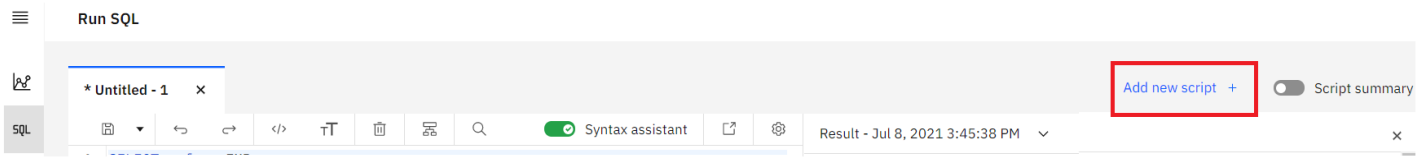
6. The SQL editor will open where you can start typing and running queries.



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



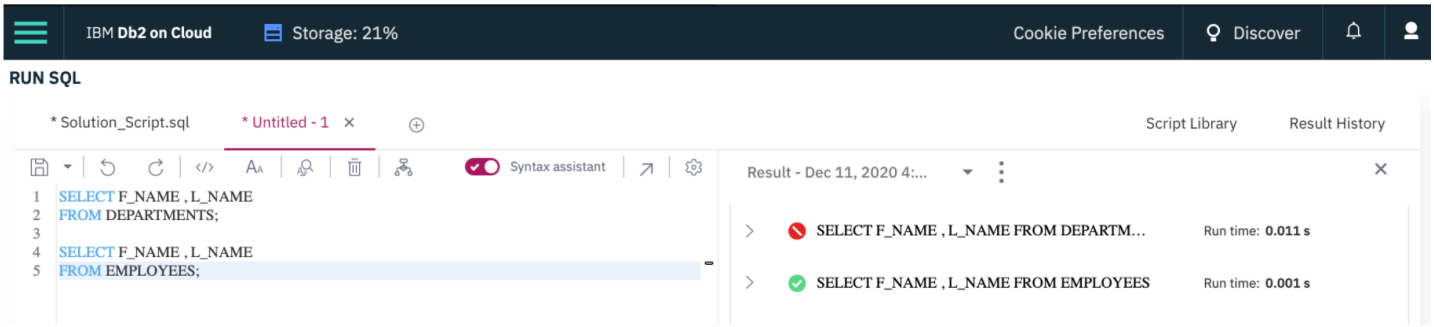
8. Click on the **Add New Script +** icon if you want to add a new script for composing queries.



9. Click on **Blank**.

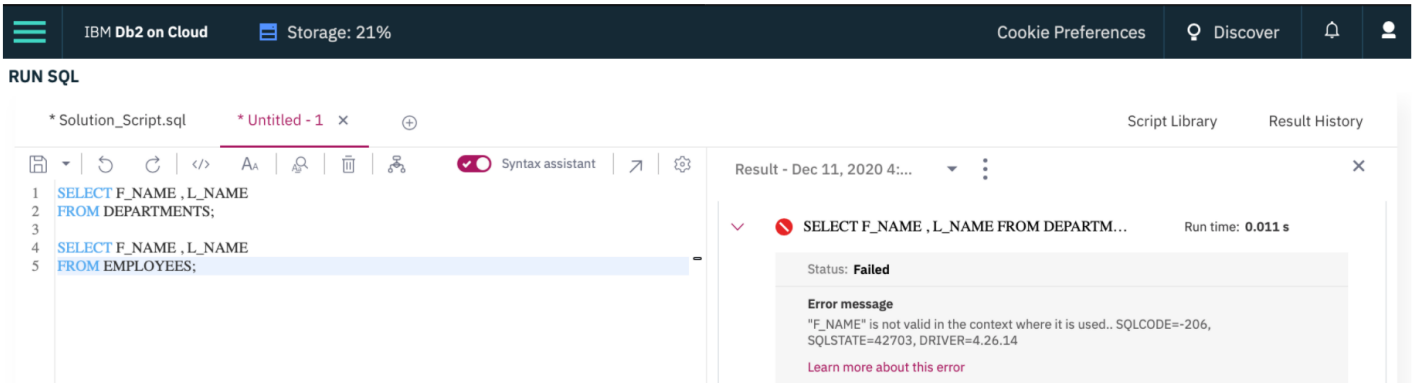
10. When you are asked in the upcoming labs, compose the appropriate SQL query for each problem and run by clicking **Run all** .

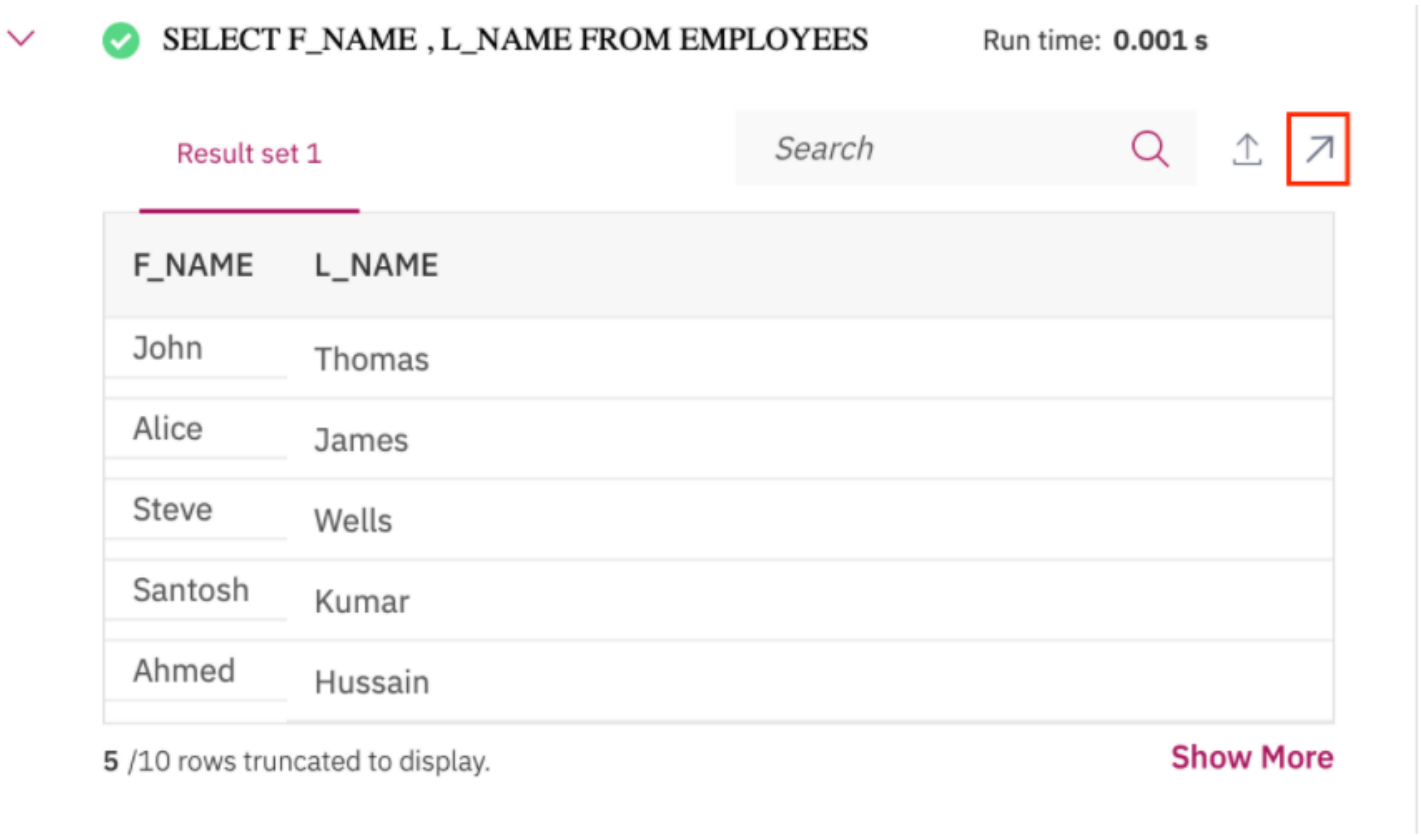
11. When you will run the script, looking at the Result section of the executed queries you will know whether the SQL statements ran successfully or not.



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.





Types of SQL Statements

Welcome to Types of SQL Statements.

At the end of the video, you will be able to distinguish between data definition language statements and data manipulation language statements.

SQL Statements are used for interacting with Entities (that is, tables),

Attributes (that is, columns) and their tuples (or rows with data values) in relational databases.

SQL statements fall into two different categories:

1. Data Definition Language statements and Data Manipulation Language statements.
2. Data Definition Language (or DDL) statements are used to define, change, or drop database objects such as tables.

Common DDL statement types include CREATE, ALTER, TRUNCATE, and DROP.

CREATE: which is used for creating tables and defining its columns;

ALTER: is used for altering tables including adding and dropping columns and modifying their datatypes;

TRUNCATE: is used for deleting data in a table but not the table itself;

DROP: is used for deleting tables.

Data Manipulation Language (or DML)

- statements are used to read and modify data in tables.
- These are also sometimes referred to as CRUD operations, that is, Create, Read, Update and Delete rows in a table.

Common DML statement types include INSERT, SELECT, UPDATE, and DELETE.

INSERT: is used for inserting a row or several rows of data into a table;

SELECT: reads or selects row or rows from a table;

UPDATE: edits row or rows in a table;

And DELETE: removes a row or rows of data from a table.

Now you know that:

DDL or Data Definition Language statements are used for defining or changing objects in a database such as tables.

And DML or Data Manipulation Language statements are used for manipulating or working with data in tables.

Types of SQL statements (DDL vs. DML)

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

- Distinguish between Data Definition Language statements and Data Manipulation Language statements

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

Types of SQL Statements - DML

- 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

Hello, and welcome to the CREATE TABLE Statement.

In this video, we will learn how to create a relational database table.

At the end of the video, you will be able to distinguish between

data definition language statements and data manipulation language statements,

and explain how the entity name and attributes

are used to create a relational database table.

SQL Statements are used for interacting with Entities (that is Tables),

Attributes (that is Columns) and their tuples (or rows with data values) in relational databases.

SQL statements fall into two different categories:

Data Definition Language statements, and Data Manipulation Language statements.

Data Definition Language (or DDL) statements are used to define, change, or drop database objects such as tables.

Common DDL statement types include: CREATE, ALTER, TRUNCATE and DROP.

CREATE: which is used for creating tables and defining its columns

ALTER: is used for altering tables including adding and dropping columns and modifying their datatypes

TRUNCATE: is used for deleting data in a table but not the table itself

DROP: is used for deleting tables

Data Manipulation Language (or DML) statements are used to read and modify data in tables.

These are also sometimes referred to as CRUD operations,

that is, Create, Read, Update and Delete rows in a table.

Common DML statement types include: INSERT, SELECT, UPDATE and DELETE .

INSERT: is used for inserting a row or several rows of data into a table

SELECT: reads or selects row or rows from a table

UPDATE: edits row or rows in a table

And DELETE: removes a row or rows of data from a table

Now let’s look at the most common DDL statement – CREATE.

The syntax of the CREATE table is shown here:

You start with “CREATE TABLE” followed by the name of the table you want to create

Then enclose rest of the statement inside a pair of parenthesis or round brackets.

Each row inside the parenthesis specifies the name of a column followed by its datatype

and possibly some additional optional values that we will see later.

Each attribute or column definition is separated by a comma.

For example, if we want to create a table for provinces in Canada you would specify:

CREATE TABLE provinces (id char(2) Primary key not null comma name varchar(24) close parenthesis

In this example, the data types used are: CHAR which is a character string of a fixed length, in this case 2.

And VARCHAR, which is a character string of a variable length.

In this case, this variable character field can be up to 24 characters long.

Issuing this statement would create a table in the database with 2 columns.

The first column id for storing the abbreviated 2 letter province short codes such as AB , BC, etc.

And the second column called name for storing the full name of the province, such as ALBERTA, BRITISH COLUMBIA, etc.

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))

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.

Now you know that:

DDL or Data Definition Language statements are used for defining or changing objects

in a database such as tables.

DML or Data Manipulation Language statements are used for manipulating or working with data in tables.

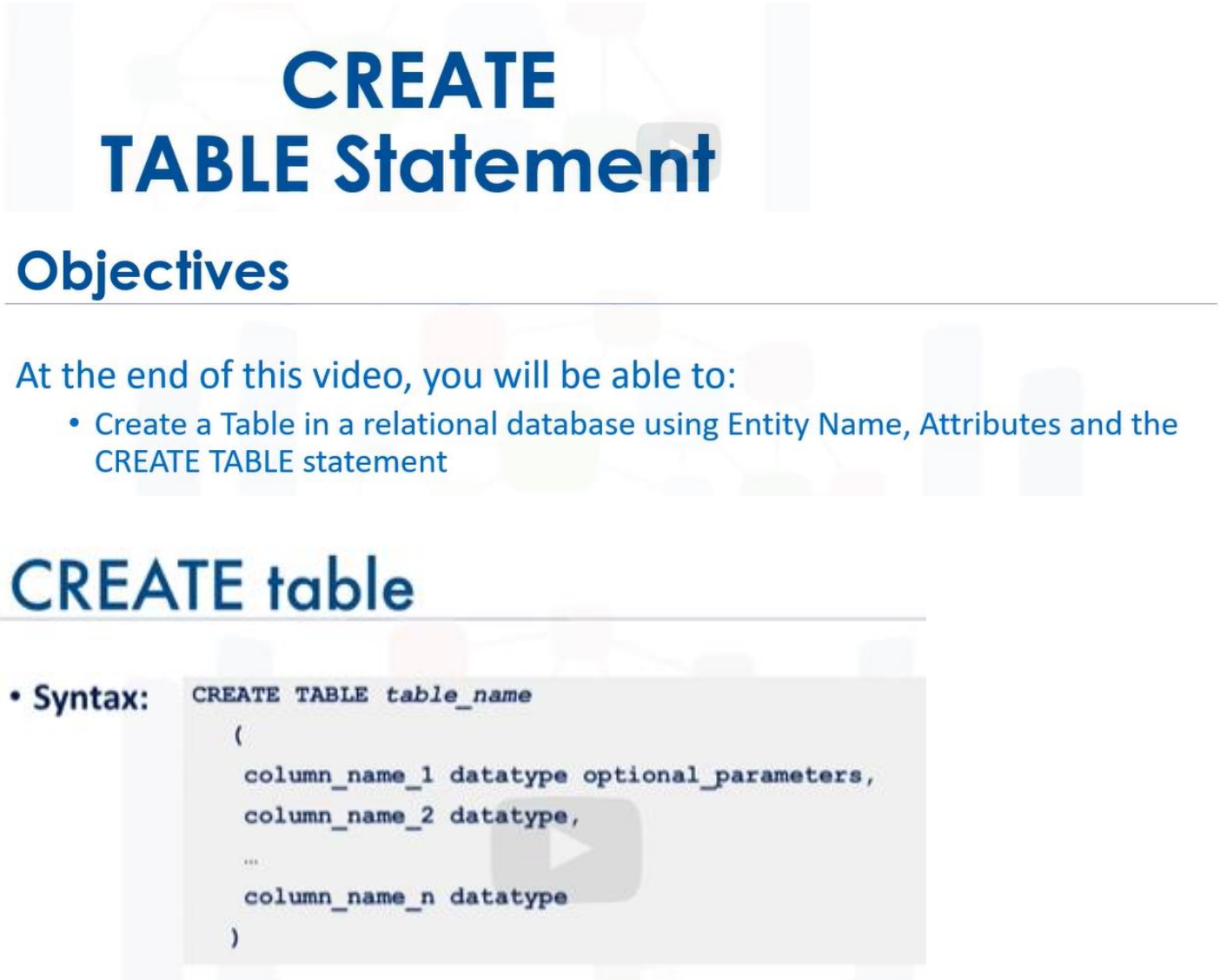
CREATE is a DDL statement for creating Entities or tables in a database.

The CREATE TABLE statement includes definition of attributes of columns in the table, including

Names of columns

Datatypes of columns

And other Optional values if required such as the Primary Key constraint



EXAMPLE

- Create a table for Canadian provinces

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

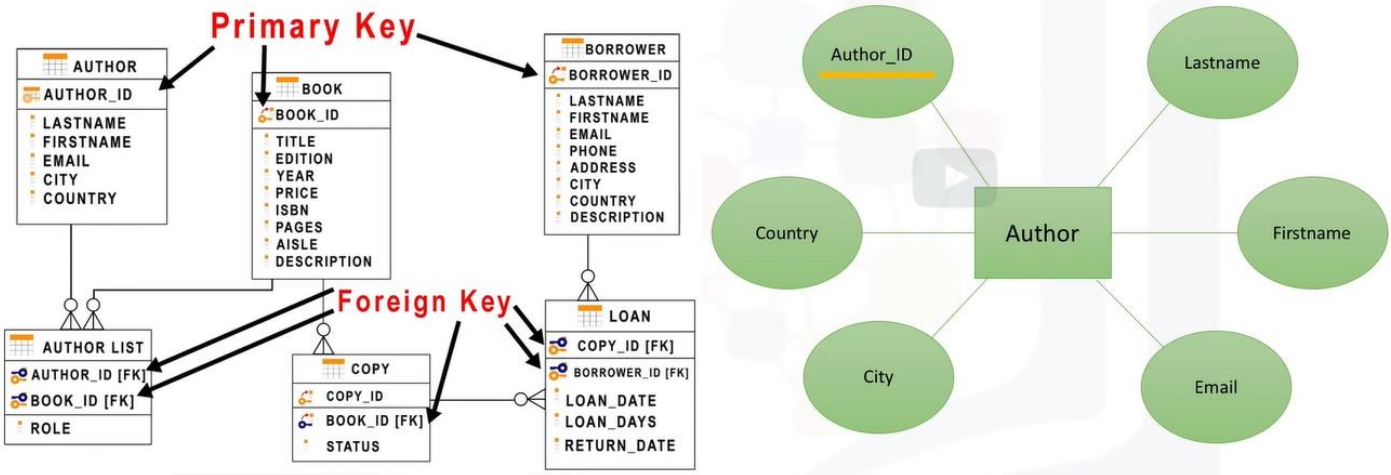
EXAMPLE

- 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

CREATE TABLE Statement

To create the Author table, use the following columns and datatypes:

AUTHOR(Author_ID:char, Lastname:varchar, Firstname:varchar, Email:varchar, City:varchar, Country:char)

```
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 Statement

To create the Author table, use the following columns and datatypes:

AUTHOR(Author_ID:char, Lastname:varchar, Firstname:varchar, Email:varchar, City:varchar, Country:char)

```
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)
   )
```

Summary

Now you know that:

- CREATE used for creating entities (tables) in a relational database
- CREATE TABLE statement includes definition of attributes (columns):
  - Names of columns
  - Datatypes of columns
  - Constraints (e.g. Primary Key)

ALTER, DROP, and Truncate Tables

Hello, and welcome to ALTER, DROP, and TRUNCATE tables.

After watching this video, you will be able to:

Describe the ALTER TABLE, DROP TABLE, and TRUNCATE statements.

Explain the syntax.

Use the statements in queries.

You use the ALTER TABLE statement to add or remove columns from a table, to modify the data type of columns, to add or remove keys, and to add or remove constraints.

The syntax of the ALTER TABLE statement is shown here.

You start with ALTER TABLE followed by the name of the table that you want to alter.

Differently to the CREATE TABLE statement though, you do not use parentheses to enclose the parameters for the ALTER TABLE statement.

Each row in the ALTER TABLE statement specifies one change that you want to make to the table.

For example, to add a telephone number column to the AUTHOR table in the Library database to store the author's telephone number, use the following statement:

ALTER TABLE author ADD COLUMN telephone_number BIGINT;

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

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.

For example, 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.

This code shows how to alter the author table:

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

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 as shown:

ALTER TABLE author DROP COLUMN telephone_number;

Similar to using DROP COLUMN to delete a column from a table, you use the DROP TABLE statement

to delete a table from a database.

If you delete a table that contains data, by default the data will be deleted alongside the table.

The syntax for the DROP TABLE statement is:

DROP TABLE table_name ;

So, you use this statement:

DROP TABLE author;

to remove the table from the database.

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;

In this video, you learned that:

The ALTER TABLE statement changes the structure of an existing table, for example, to add, modify, or drop columns.

The DROP TABLE statement deletes an existing table.

The TRUNCATE TABLE statement deletes all rows of data in a table.

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 ... ADD COLUMN

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

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

ALTER TABLE ... ALTER 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

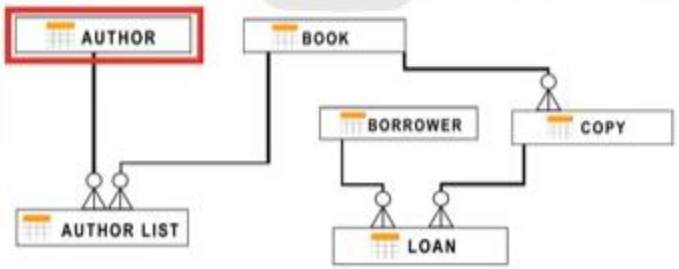
ALTER TABLE ... ALTER COLUMN

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;
```



TRUNCATE TABLE

```
TRUNCATE TABLE <table_name>
    IMMEDIATE;
```

TRUNCATE TABLE

```
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

Summary

In this video, you learned that:

- The ALTER TABLE statement changes the structure of an existing table, for example to add, modify, or drop columns
- The DROP TABLE statement deletes an existing table
- The TRUNCATE TABLE statement deletes all rows of data in a table

Reading: Examples to CREATE and DROP tables

Objective(s)

At the end of this lab you will be able to:

- Create and Drop tables in the database

Here we will look at some examples to create and drop tables. In the previous video we saw the general syntax to create a table: create table TABLENAME (COLUMN1 datatype, COLUMN2 datatype, COLUMN3 datatype, ...);

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

```
create table TEST (
    ID integer,
    NAME varchar(30)
    );
```

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

```
create table COUNTRY (
    ID int,
    CCODE char(2),
    NAME varchar(60)
    );
```

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

```
create table COUNTRY (
    ID int NOT NULL,
    CCODE char(2),
    NAME varchar(60),
    PRIMARY KEY (ID)
    );
```

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

in a table, and using Primary Keys can help speed up your queries significantly. If the table you are trying to create already exists in the database, you will get an error indicating table XXX.YYY already exists. To circumvent this error, either create a table with a different name or first DROP the existing table. It is quite common to issue a DROP before doing a CREATE in test and development scenarios. Here is an example:

```
drop table COUNTRY;
create table COUNTRY (
    ID integer PRIMARY KEY NOT NULL,
    CCODE char(2),
    NAME varchar(60)
    );
```

WARNING: before dropping a table ensure that it doesn't contain important data that can't be recovered easily. Note that if the table does not already exist and you try to drop it, you will see an error like **XXX.YYY is an undefined name**. You can ignore this error as long as the subsequent CREATE statement executed successfully.

In the lab later in this module you will practice creating tables and other SQL statements hands-on.

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

In this lab, you will learn some commonly used DDL (Data Definition Language) statements of SQL. First you will learn the CREATE statement, which is used to create a new table in a database. Next, you will learn the ALTER statement which is used to add, delete, or modify columns in an existing table. Then, you will learn the TRUNCATE statement which is used to remove all rows from an existing table without deleting the table itself. Lastly, you will learn the DROP statement which is used to delete an existing table in a database.

How does the syntax of a CREATE statement look?

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

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;
```

Software Used in this Lab

In this lab, you will use IBM Db2 Database. Db2 is a Relational Database Management System (RDBMS) from IBM, designed to store, analyze and retrieve the data efficiently.

To complete this lab you will utilize a Db2 database service on IBM Cloud. If you did not already complete this lab task earlier in this module, you will not yet have access to Db2 on IBM Cloud, and you will need to follow this lab first:

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

Database Used in this Lab

The databases used in this lab are internal databases.

Objectives

After completing this lab, you will be able to:

- Create a new table in a database
- Add, delete, or modify columns in an existing table
- Remove all rows from an existing table without deleting the table itself
- Delete an existing table in a database

Instructions

When you approach the exercises in this lab, follow the instructions to run the queries on Db2:

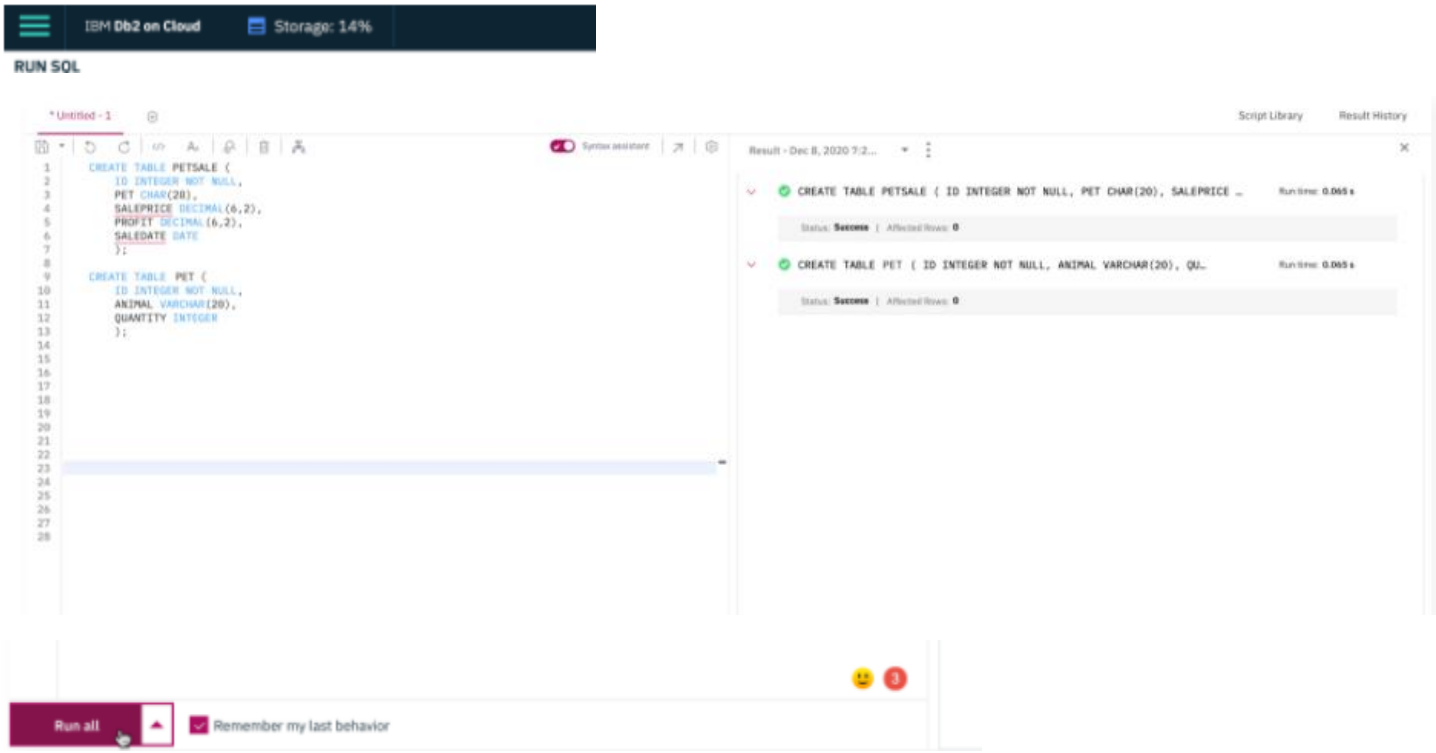
- Go to the Resource List of IBM Cloud by logging in where you can find the Db2 service instance that you created in a previous lab under **Services** section. Click on the **Db2-xx service**. Next, open the Db2 Console by clicking on **Open Console** button. Click on the 3-bar menu icon in the top left corner and go to the **Run SQL** page. The Run SQL tool enables you to run SQL statements.
    - If needed, follow Hands-on Lab : Sign up for IBM Cloud, Create Db2 service instance and Get started with the Db2 console

Exercise 1: CREATE

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

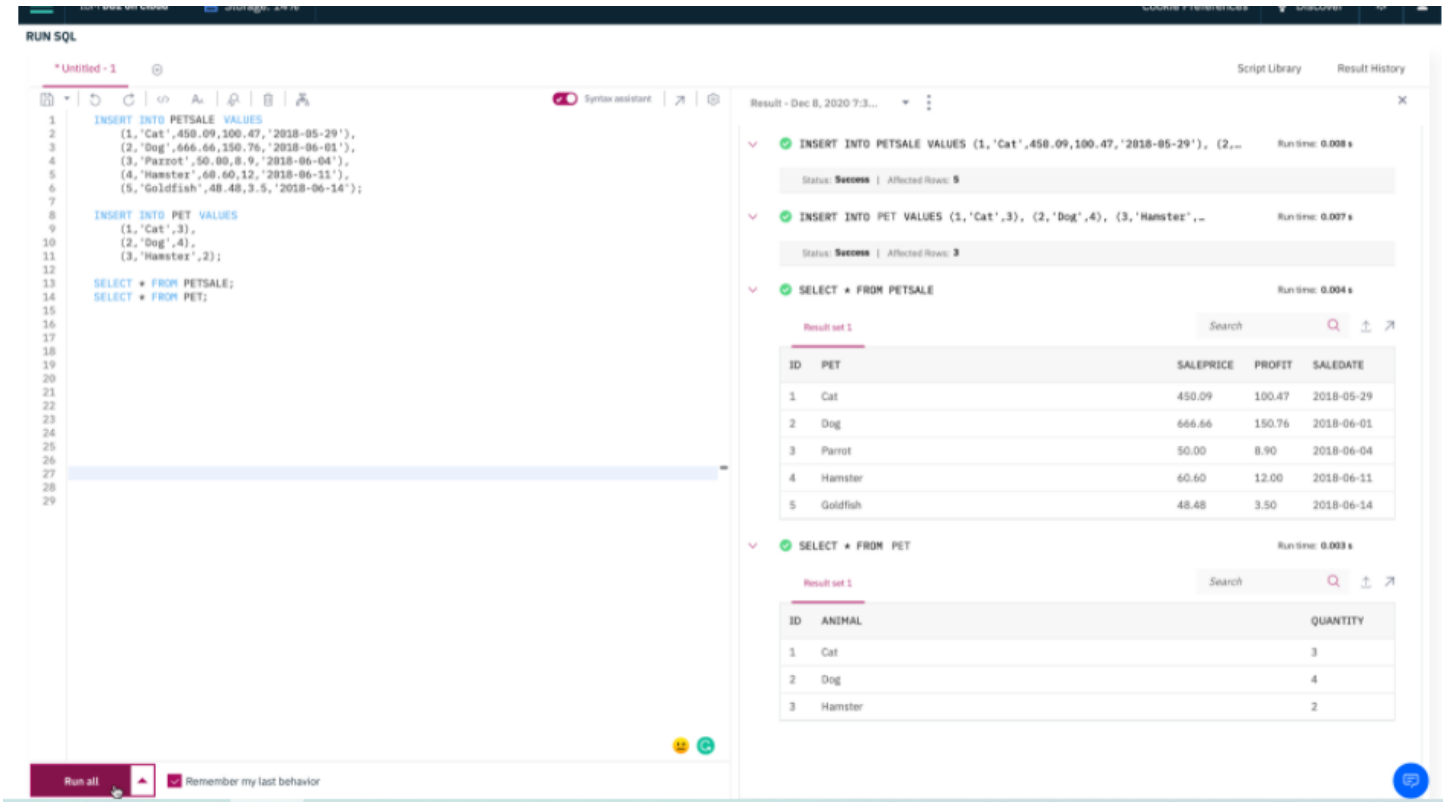
1. You need to create two tables, **PETSALE** and **PET**. To create the two tables PETSALE and PET, copy the code below and paste it to the textbox of the **Run SQL** page. Click **Run all**.

```
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;
```



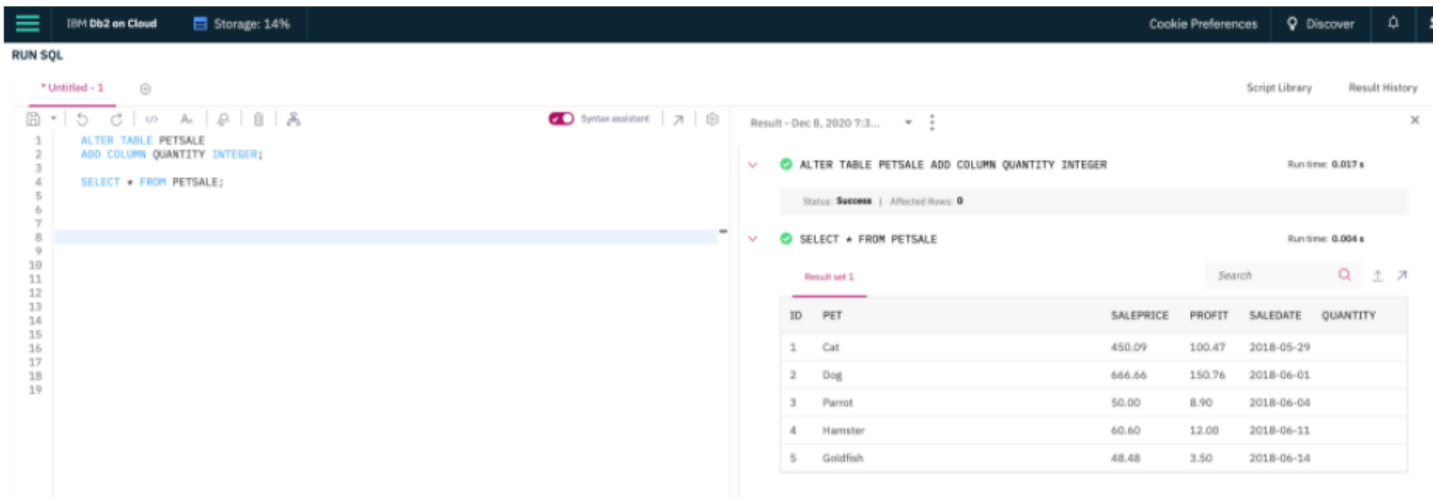
Exercise 2: ALTER

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

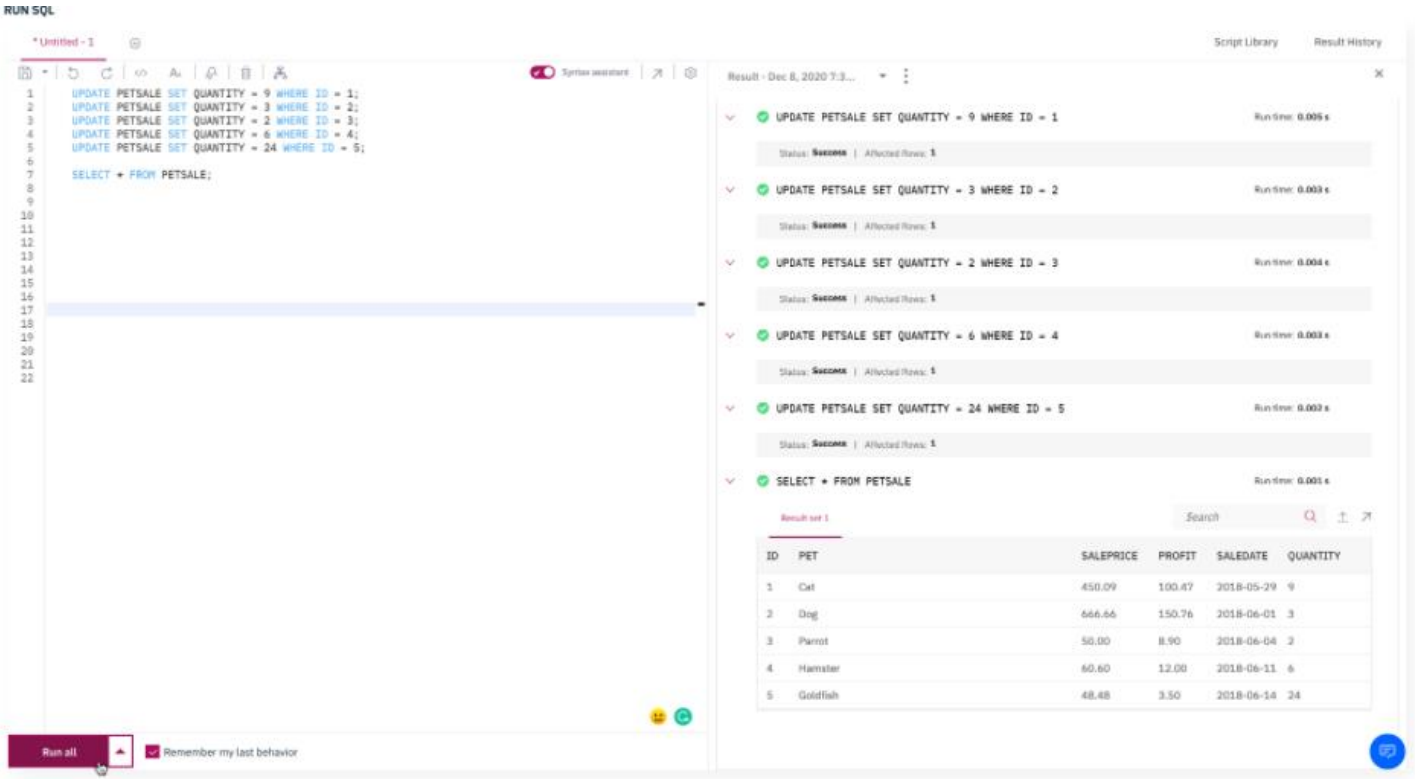
1. Add a new **QUANTITY** column to the **PETSALE** table and show the altered table. Copy the code below and paste it to the textbox of the **Run SQL** page. Click **Run all**.

```
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**.

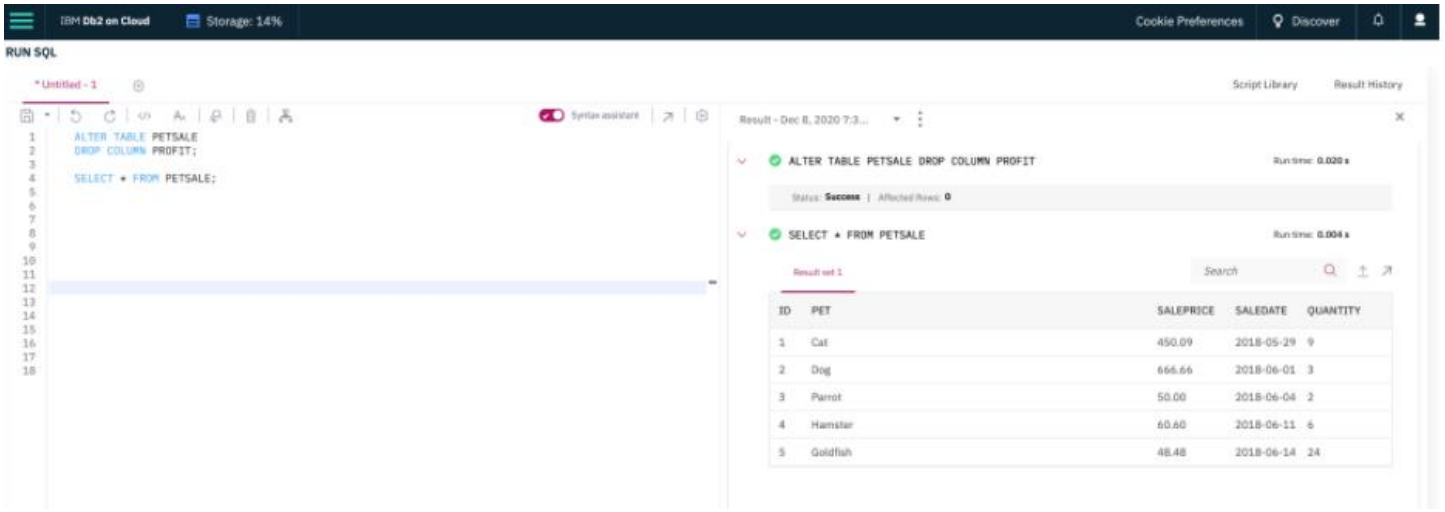
```
UPDATE PETSALE SET QUANTITY = 9 WHERE ID = 1;
UPDATE PETSALE SET QUANTITY = 3 WHERE ID = 2;
UPDATE PETSALE SET QUANTITY = 2 WHERE ID = 3;
UPDATE PETSALE SET QUANTITY = 6 WHERE ID = 4;
UPDATE PETSALE SET QUANTITY = 24 WHERE ID = 5;

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**.

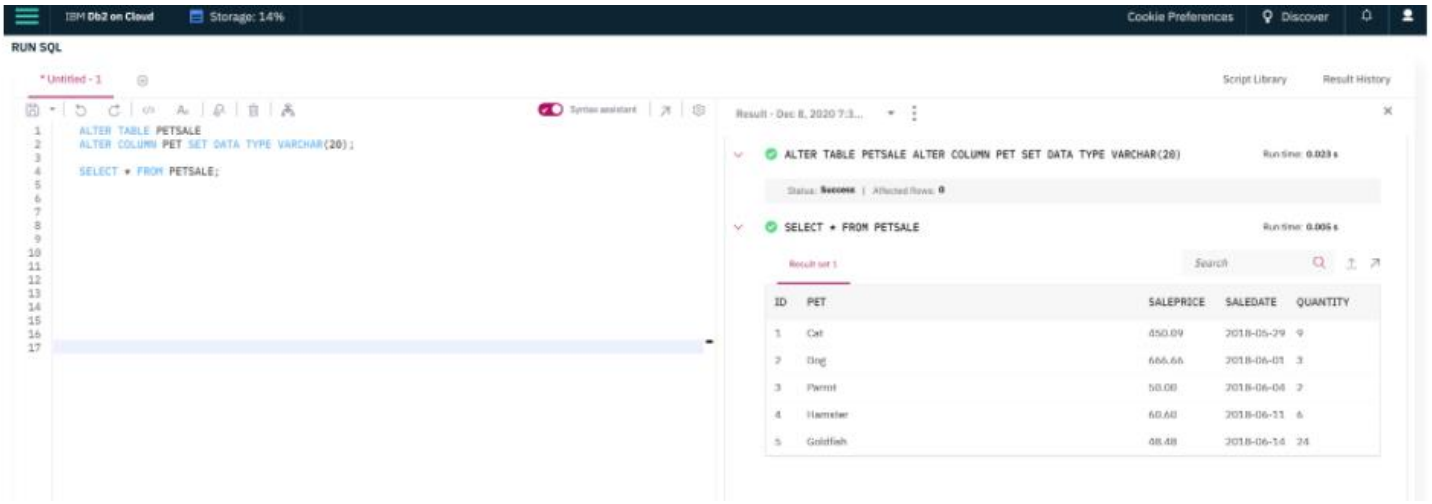
```
ALTER TABLE PETSALE
DROP COLUMN PROFIT;

SELECT * FROM PETSALE;
```



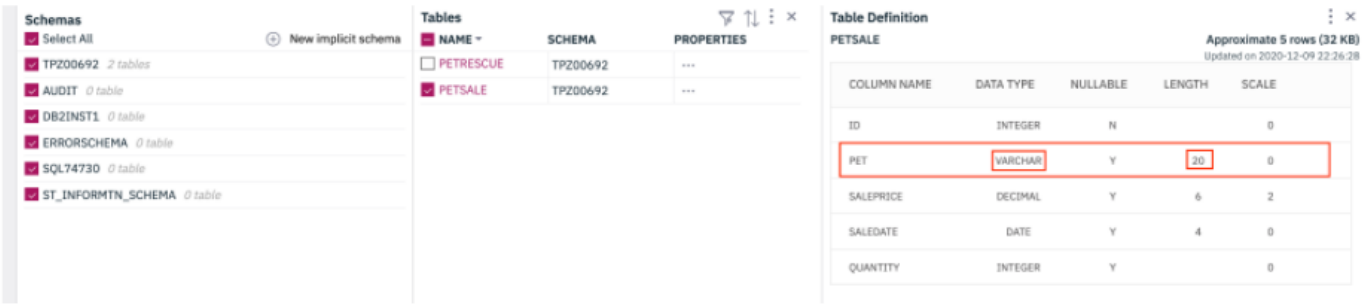
Task C: ALTER using ALTER COLUMN

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;
```



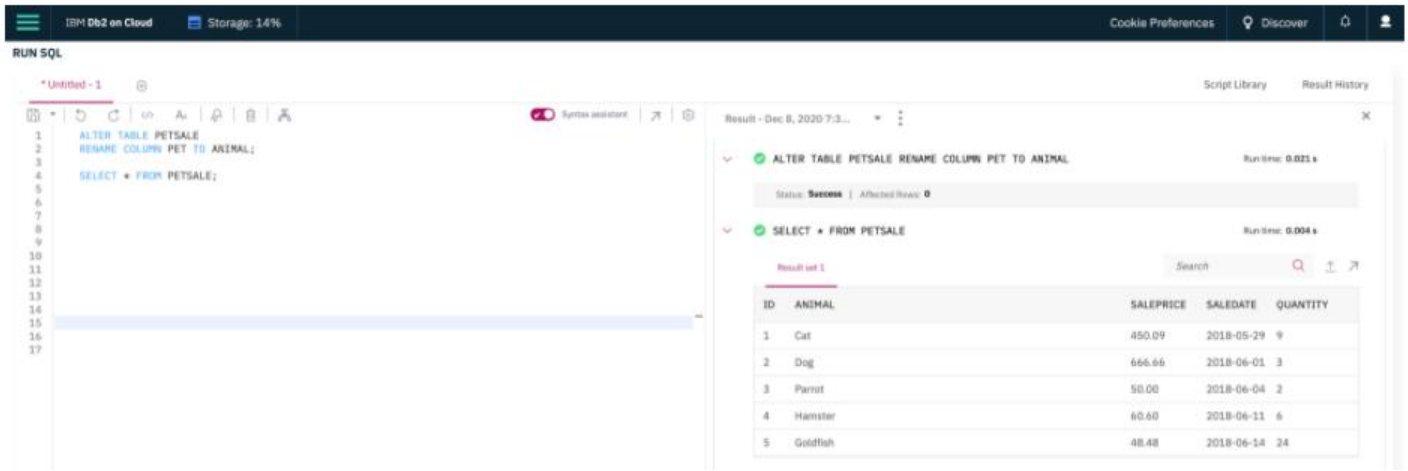
2. Now verify if the data type of the column **PET** of the table **PETSALE** changed to **VARCHAR(20)** type or not. Click on the 3 bar menu icon in the top left corner and click **Explore** > **Tables**. Find the **PETSALE** table from Schemas by clicking **Select All**. Click on the **PETSALE** table 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;
```

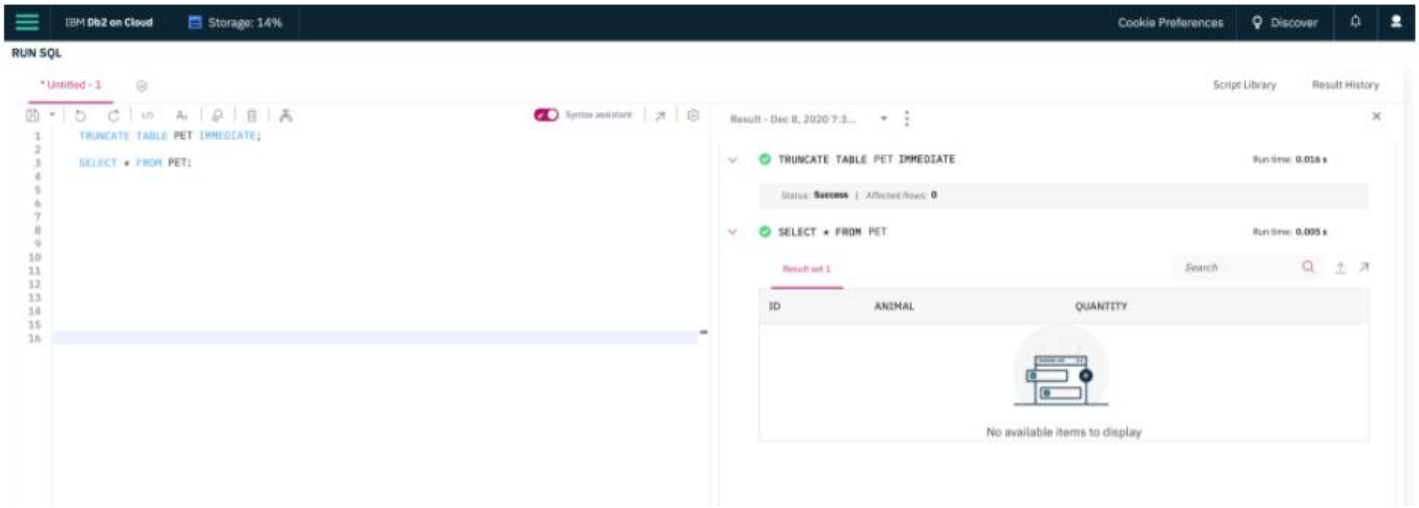


Exercise 3: TRUNCATE

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;
```

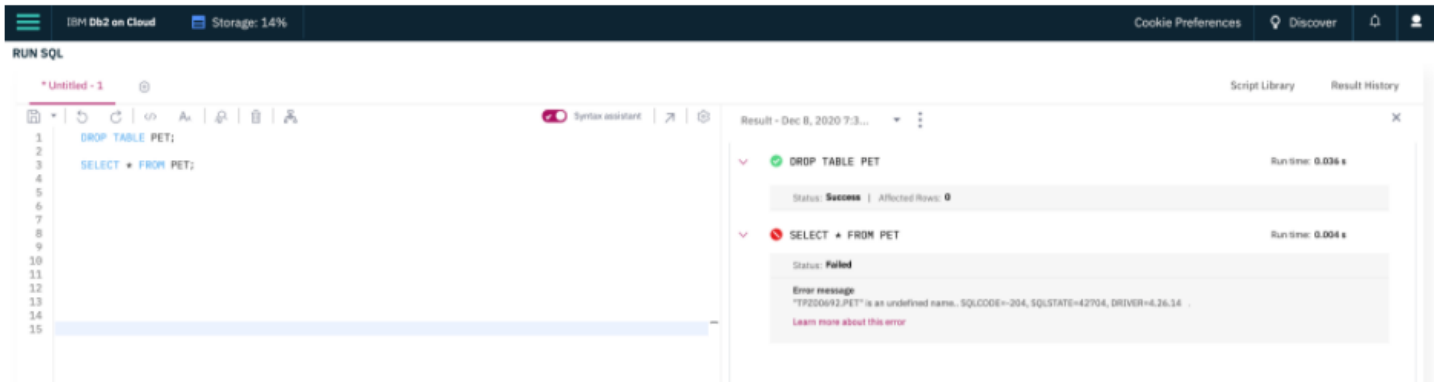


Exercise 4: DROP

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

Estimated time needed: 30 minutes

In this lab, you will learn how to run SQL scripts to create several tables at once, as well as how to load data into tables from .csv files.

Software Used in this Lab

In this lab, you will use IBM Db2 Database. Db2 is a Relational Database Management System (RDBMS) from IBM, designed to store, analyze and retrieve the data efficiently.

To complete this lab you will utilize a Db2 database service on IBM Cloud. If you did not already complete this lab task earlier in this module, you will not yet have access to Db2 on IBM Cloud, and you will need to follow this lab first:

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

Database Used in this Lab

The database used in this lab is an internal database. You will be working on a sample HR database. This HR database schema consists of 5 tables called EMPLOYEES, JOB_HISTORY, JOBS, DEPARTMENTS and LOCATIONS. Each table has a few rows of sample data. The following diagram shows the tables for the HR database:

SAMPLE HR DATABASE TABLES

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

JOB_HISTORY

EMPL_ID	START_DATE	JOBS_ID	DEPT_ID
E1001	2000-01-30	100	2
E1002	2010-08-16	200	5
E1003	2016-08-10	300	5

JOBS

JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY
100	Sr. Architect	60000	100000
200	Sr.SoftwareDeveloper	60000	80000
300	Jr.SoftwareDeveloper	40000	60000

DEPARTMENTS

DEPT_ID_DEP	DEP_NAME	MANAGER_ID	LOC_ID
2	Architect Group	30001	L0001
5	Software Development	30002	L0002
7	Design Team	30003	L0003
5	Software	30004	L0004

LOCATIONS

LOCT_ID	DEP_ID_LOC
L0001	2
L0002	5
L0003	7

Objectives

After completing this lab, you will be able to:

Create tables using SQL scripts

Load data into tables

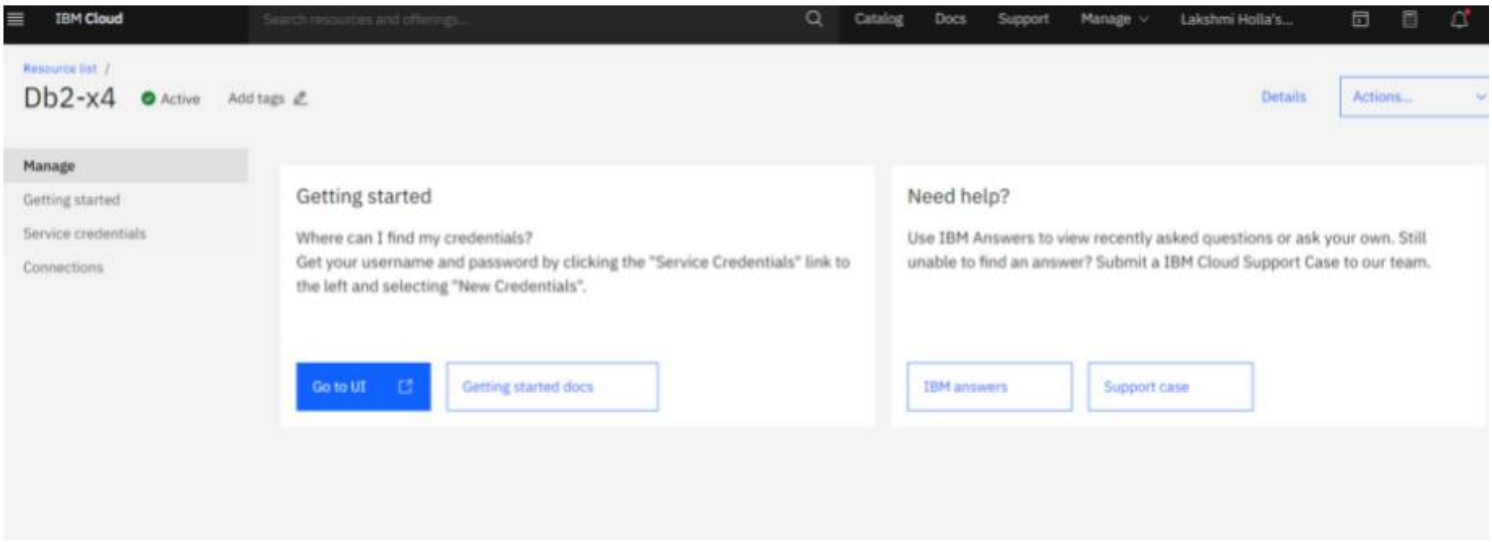
Exercise 1: Create tables using SQL scripts

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.

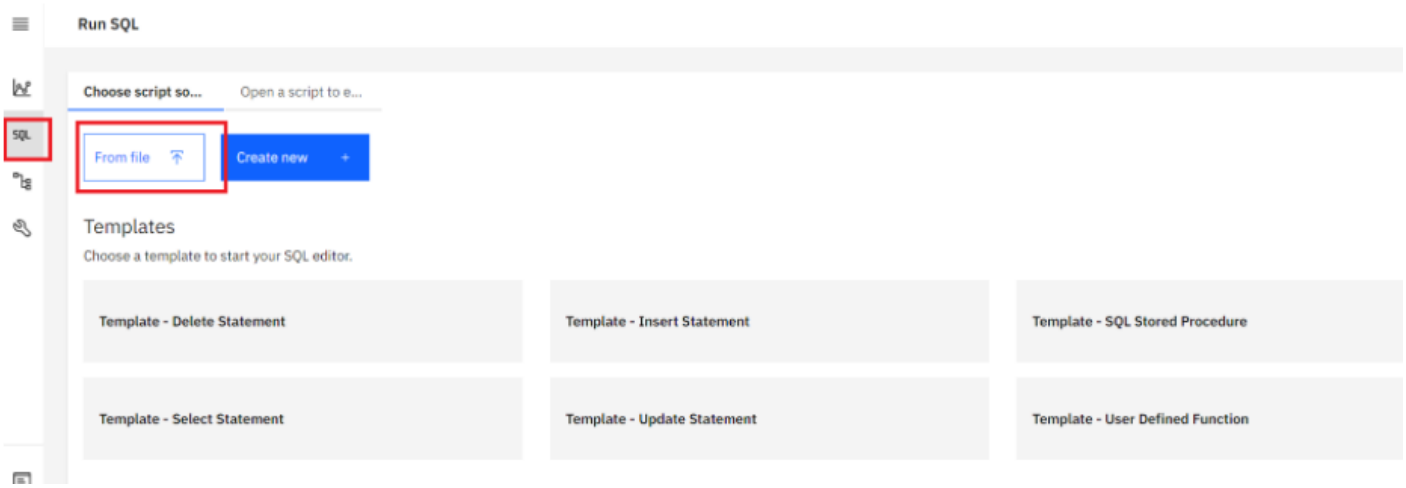
Download the script file to your computer:

HR_Database_Create_Tables_Script.sql

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.

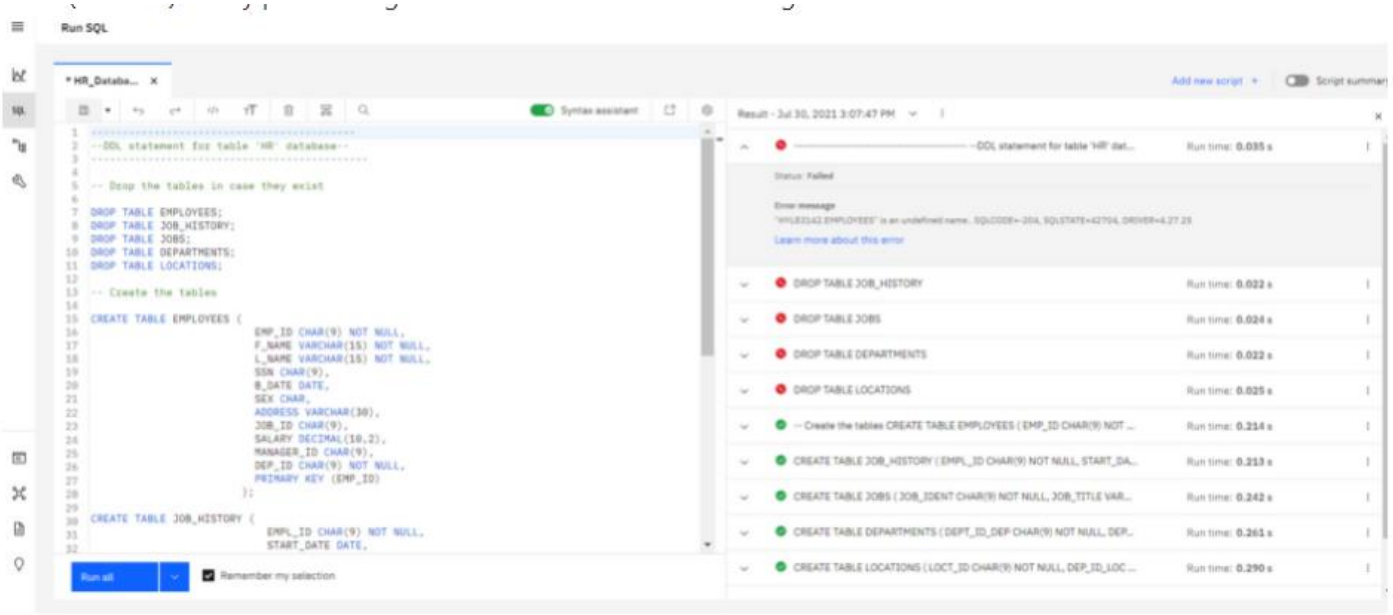


3. Click on Run SQL on the left corner and select the from file option.

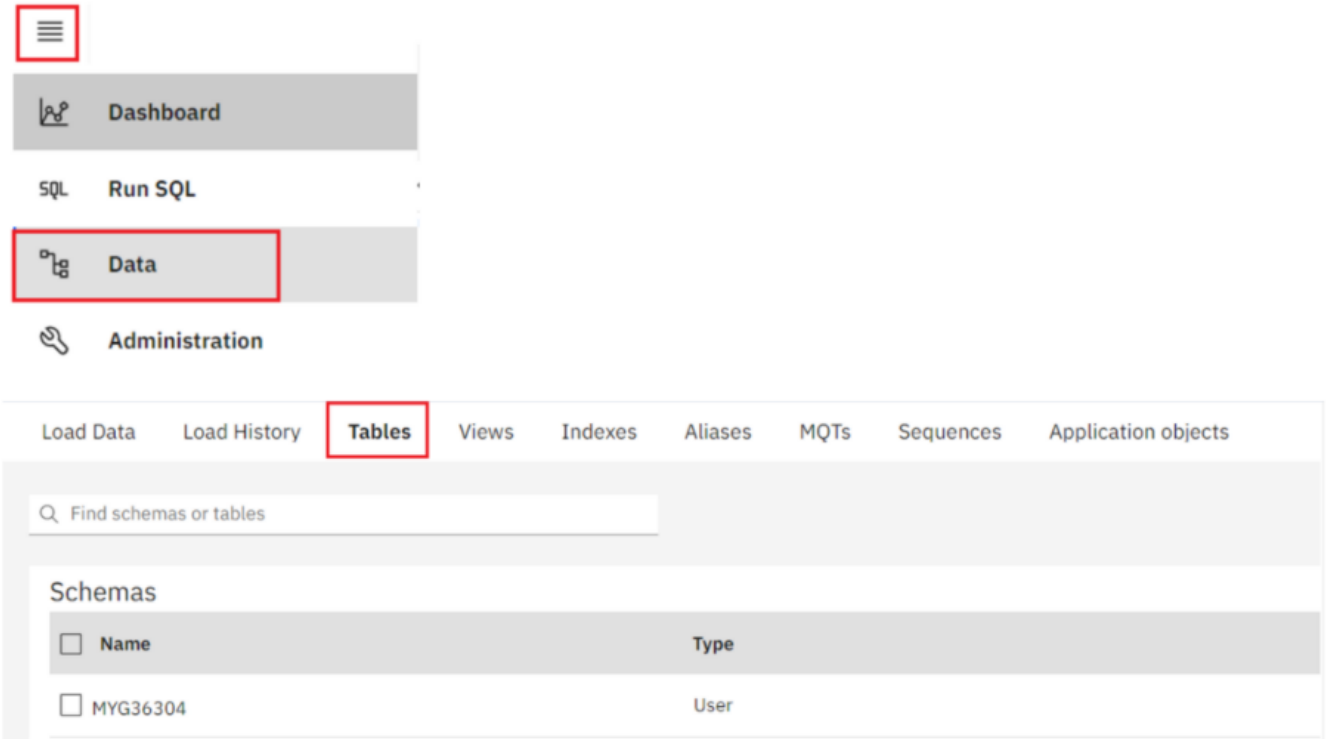


2. Locate the file **HR_Database_Create_Tables_Script.sql** that you downloaded to your computer earlier and open it.
3. Once the statements are in the SQL Editor tool , you can run the queries against the database by selecting the **Run All** button.

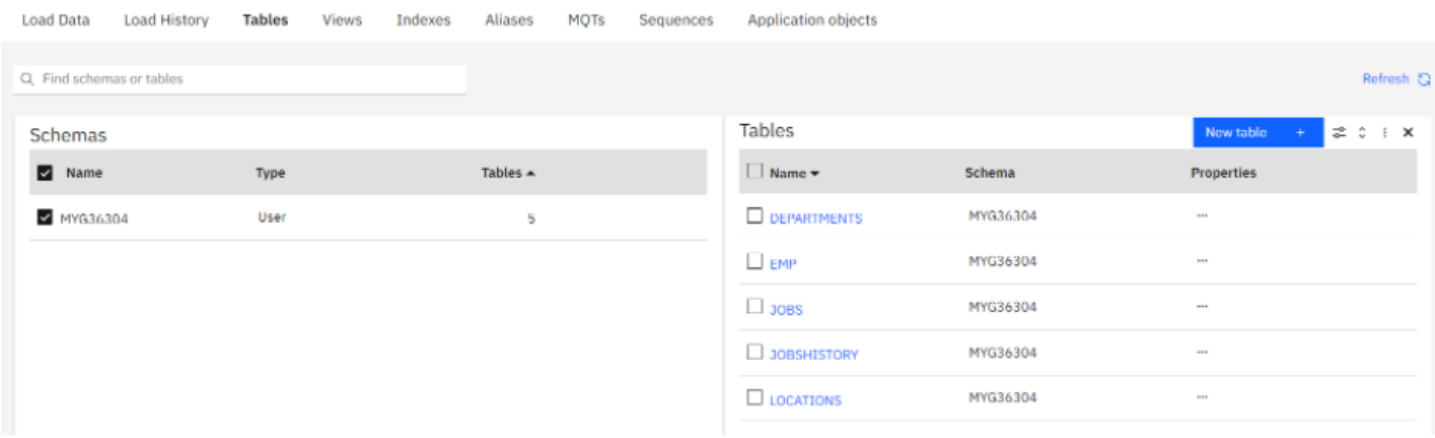
4. On the right side of the SQL editor window you will see a Result section. Clicking on a query in the Result section will show the execution details of the job like whether it ran successfully, or had any errors or warnings. Ensure your queries ran successfully and created all the tables.
    - **Note:** 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.



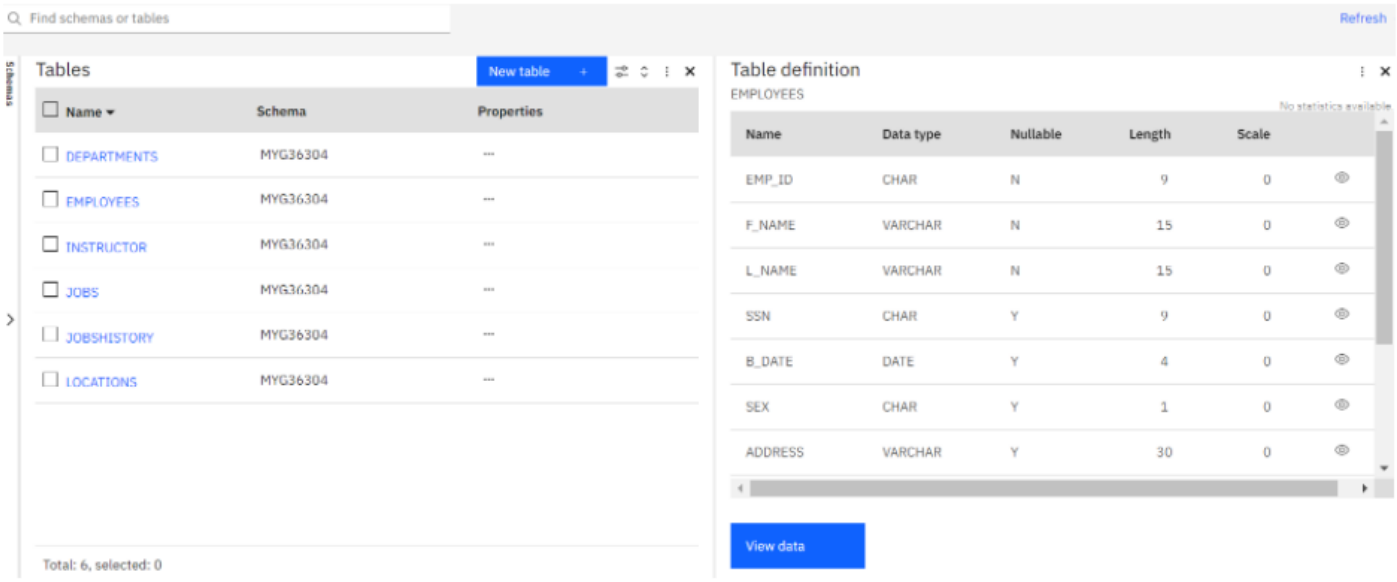
7. Now you can look at the tables you created. Click on the data icon and then click on Tables tab



2. 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).



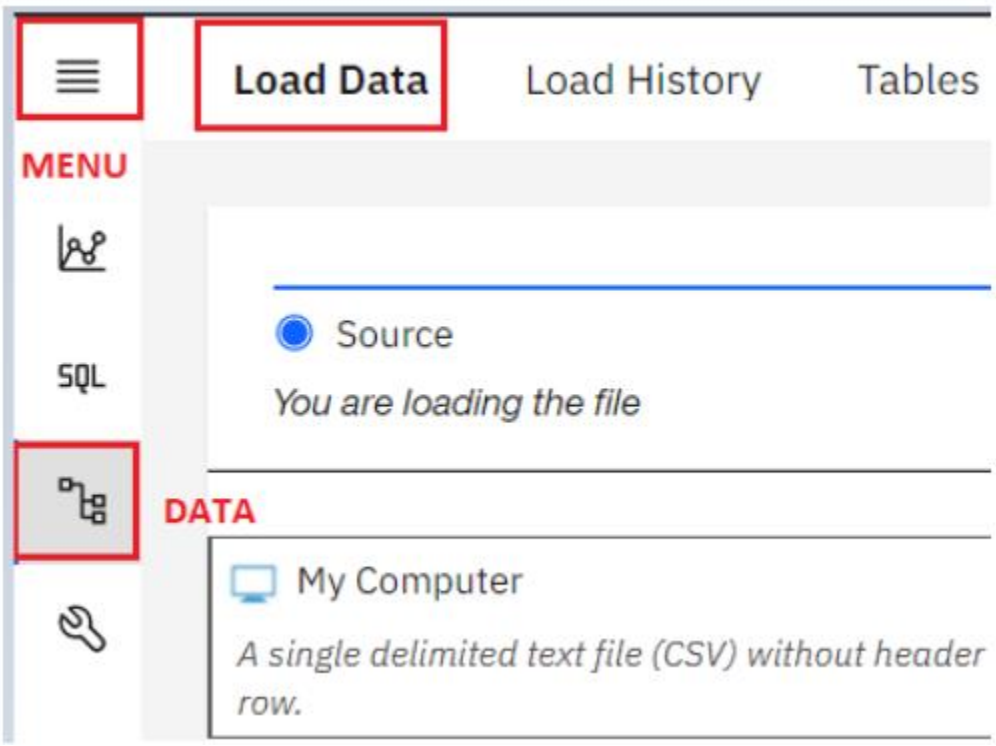
Exercise 2: Load data into tables

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.

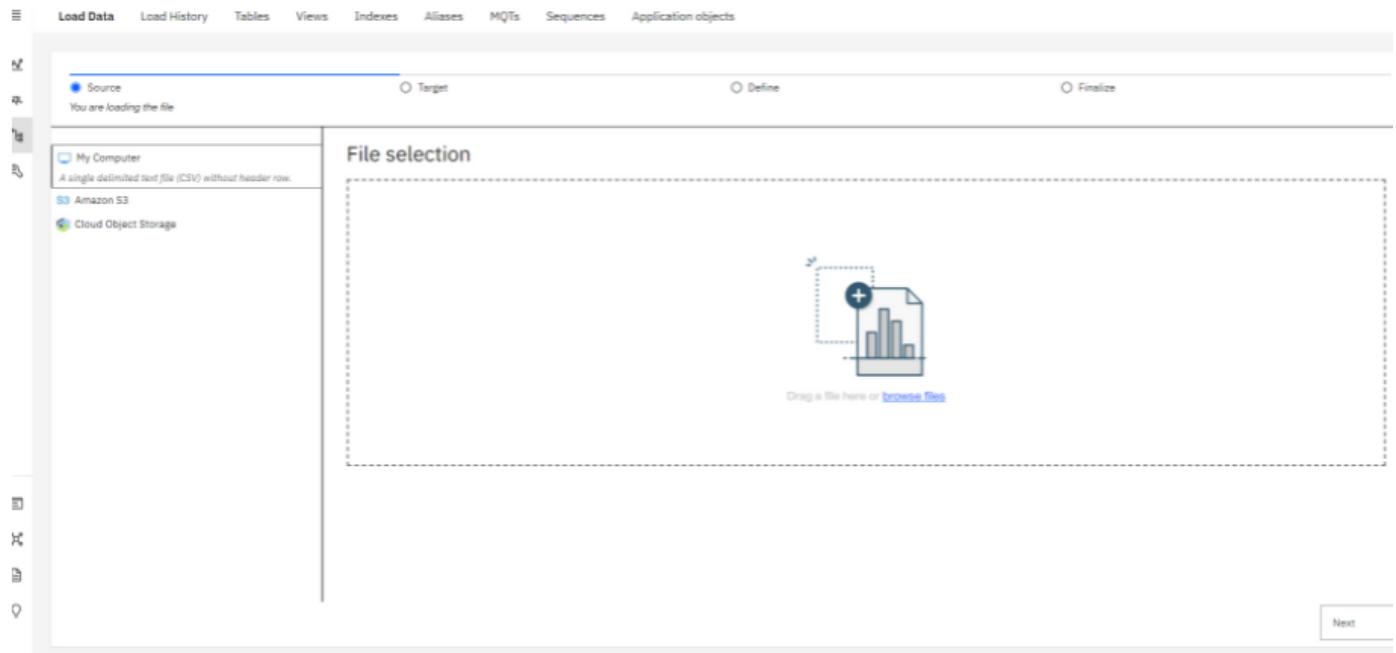
The steps below explain the process of loading data into the tables you created earlier in exercise 1.

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

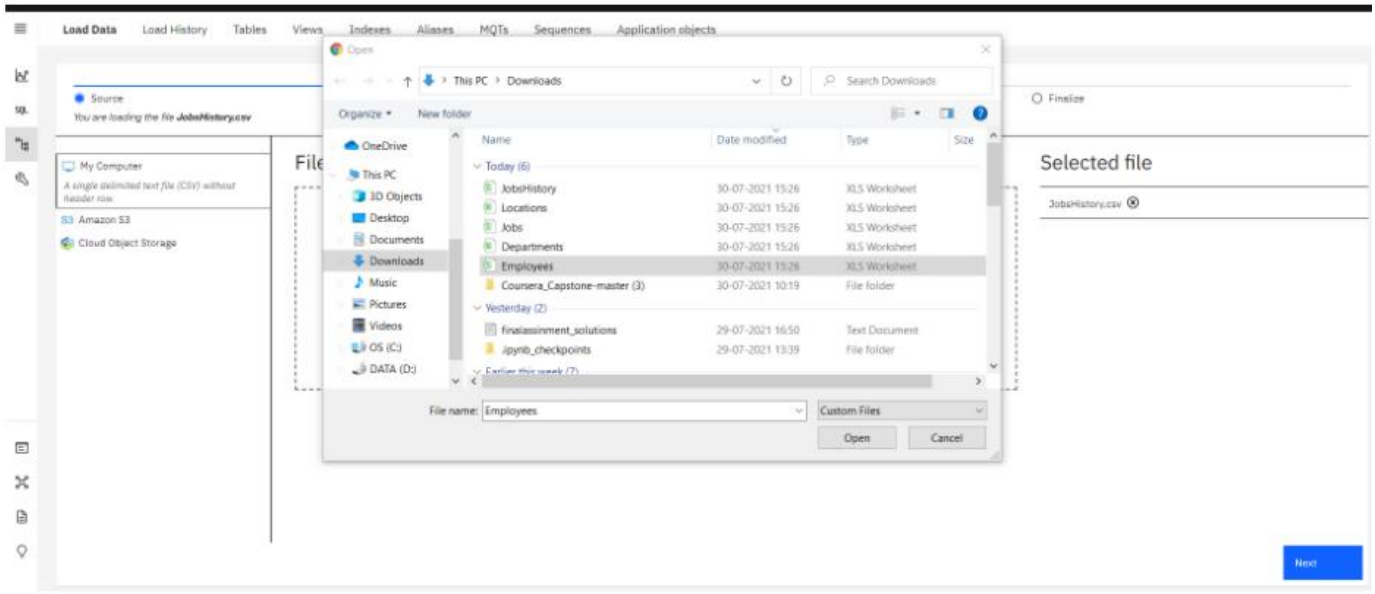
2. In the Db2 Console, from the 3-bar menu icon in the top left corner, click **Load**, and then select **Load Data**.



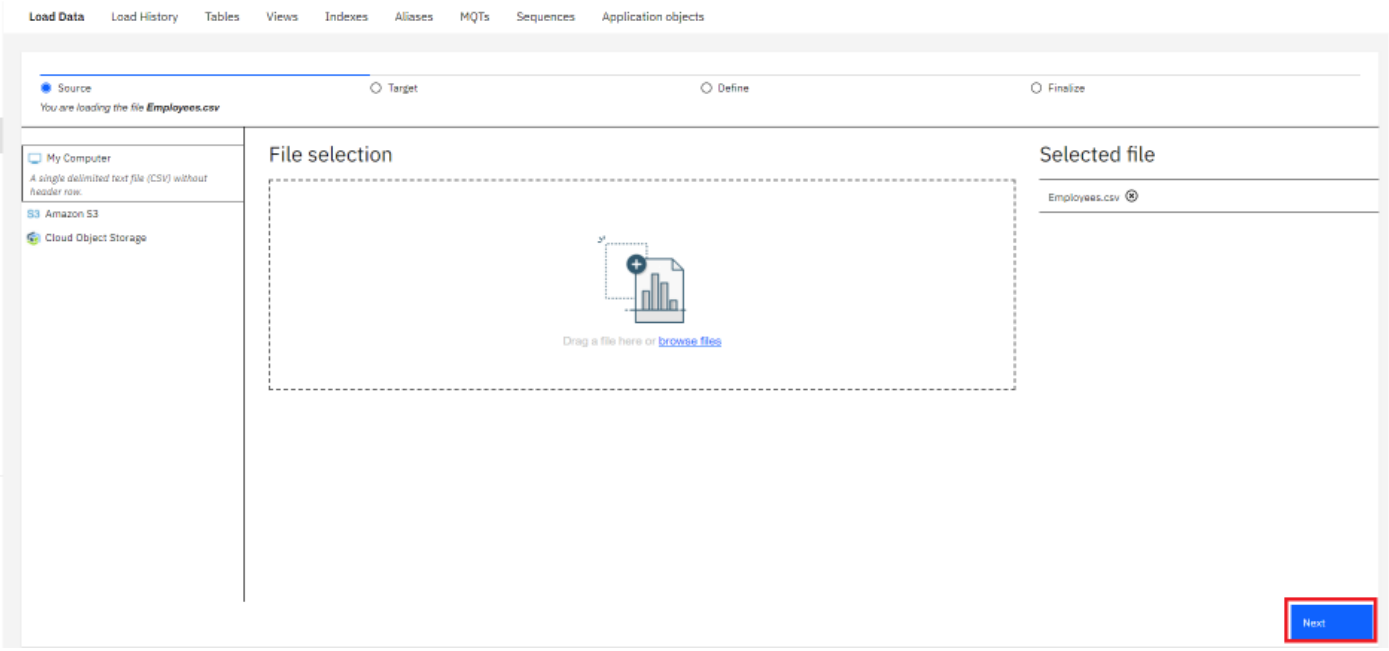
3. On the **Load Data** page that opens, ensure **My Computer** is selected as the source. Click on the **browse files** link.



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

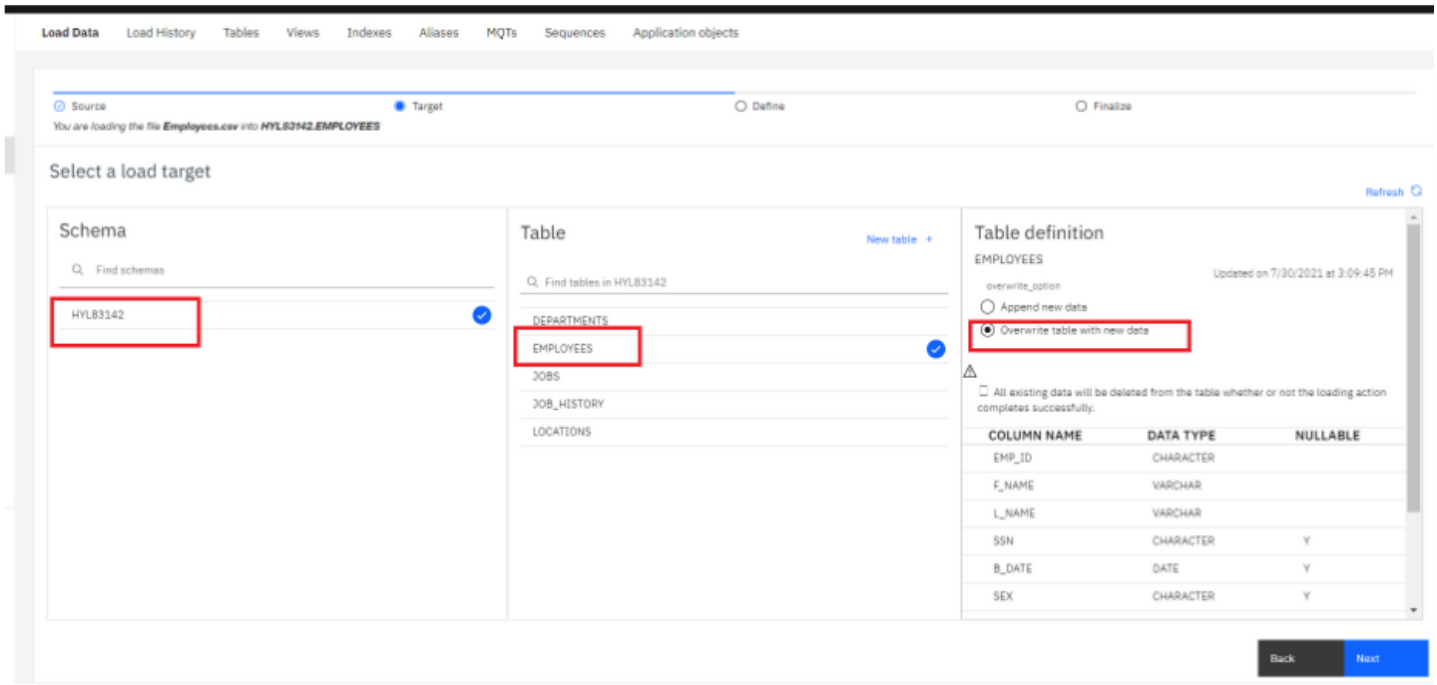


5. Once the File is selected, click **Next** in the bottom right corner.

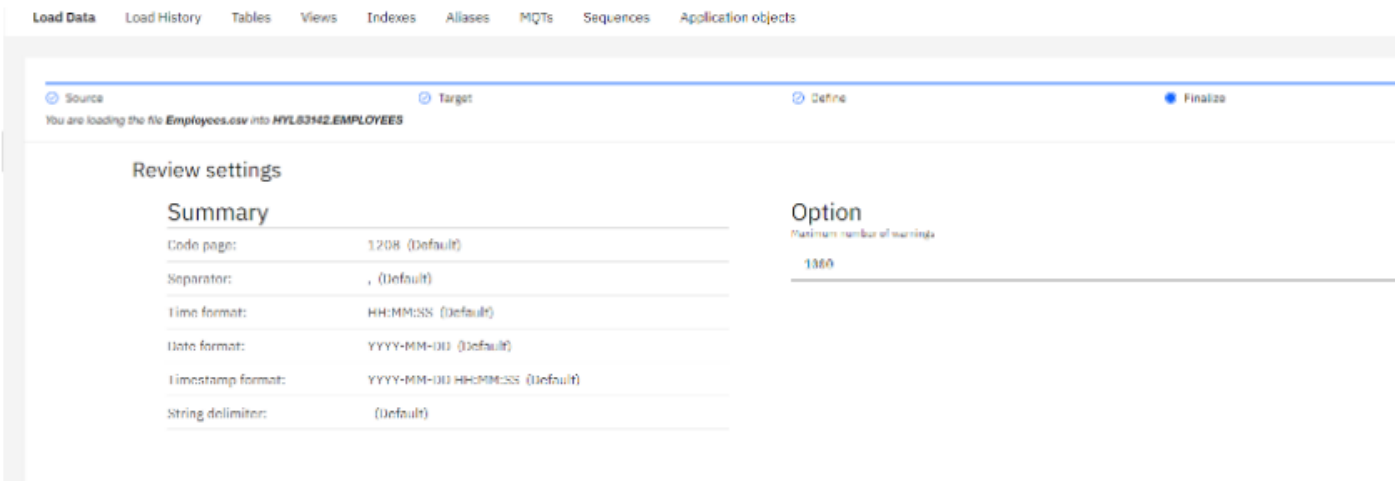


6. Select the schema for your Db2 Userid (the one where you created the tables earlier).It will show all the tables that have been created in this schema previously, including the Employees table. Select the **EMPLOYEES** table, and in the new Table Definition tab that appears, choose **Overwrite table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

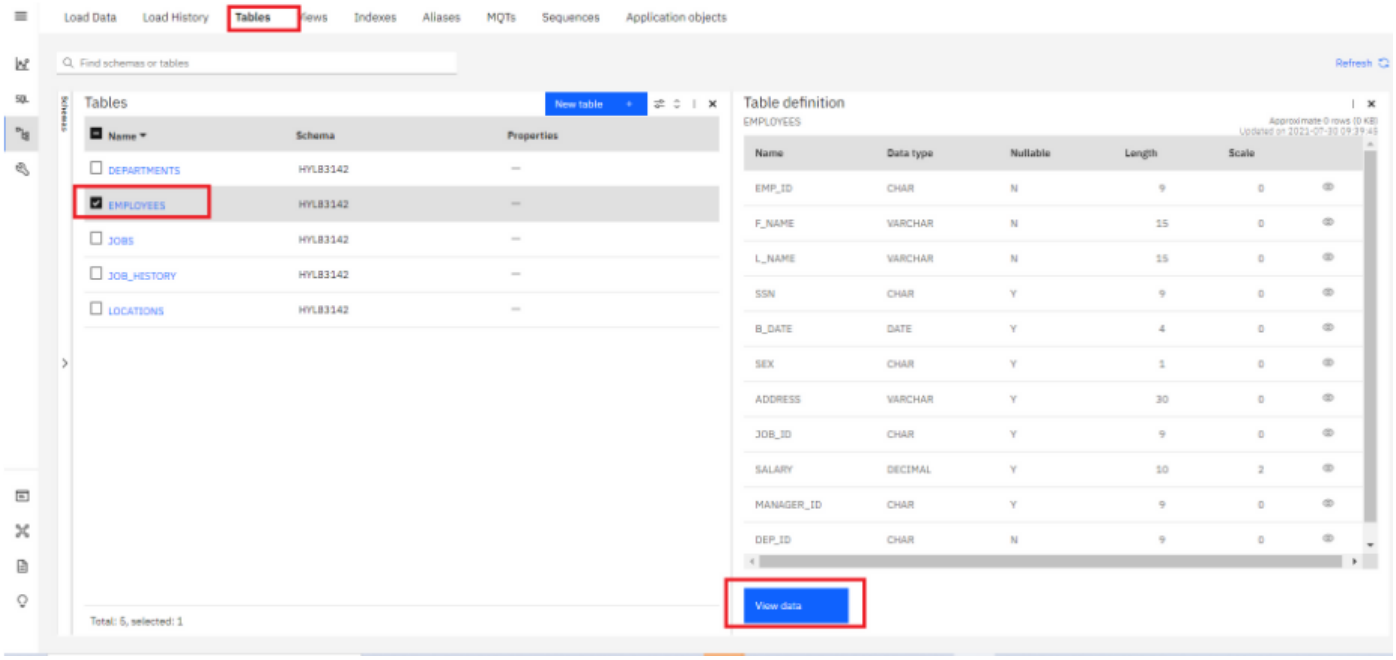


8. Click **Next**. Review the load settings and click **Begin Load** in the bottom right corner.

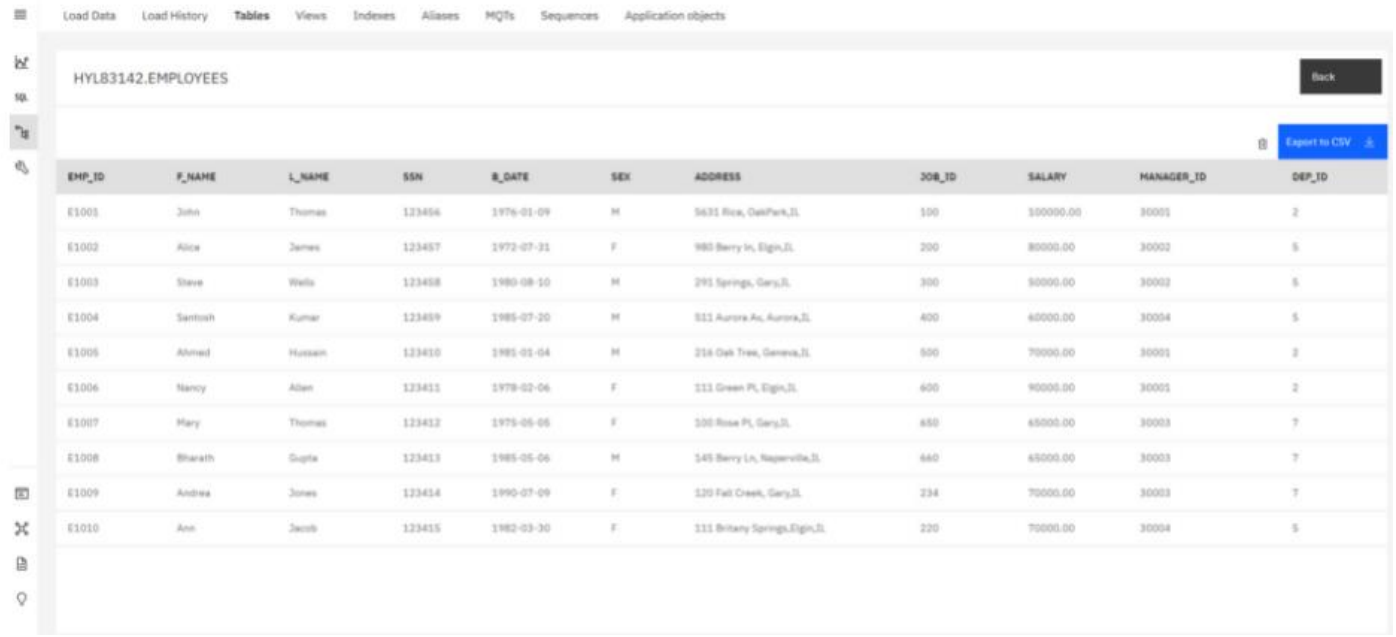


9. After loading has completed, you will notice that you were successful in loading all 10 rows of the Employees table. If there are any **Errors** or **Warnings**, you can see them on this screen.

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.

Summary & Highlights

Bookmarked

Congratulations! You have completed this lesson. At this point in the course, you know:

- A database is a repository of data that provides functionality for adding, modifying, and querying the data.
- SQL is a language used to query or retrieve data from a relational database.
- The Relational Model is the most used data model for databases because it allows for data independence.
- The primary key of a relational table uniquely identifies each tuple or row, preventing duplication of data and providing a way of defining relationships between tables.
- SQL statements fall into two different categories: Data Definition Language (DDL) statements and Data Manipulation Language (DML) statements.

Module 3 - Intermediate SQL - Introduction & Learning Objectives

In this module, you will learn how to use string patterns and ranges to search data and how to sort and group data in result sets. You will also practice composing nested queries and execute select statements to access data from multiple tables.

Learning Objectives

Demonstrate how to use string patterns and ranges in SQL queries

Demonstrate the use of grouping data in result sets

Demonstrate how to sort and order result sets

Demonstrate how to write sub-queries and nested selects

Build queries to access multiple tables

Using String Patterns, Ranges

Hello, and welcome to retrieving data with SELECT statements string patterns.

In this video, we will learn about some advanced techniques in retrieving data from a relational database table.

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. What if we wanted to retrieve the list of books whose number of pages is more than 290, but less than 300.

We could write the SELECT statement like this, specifying the WHERE clause as, where pages is greater than or equal to 290, and pages is less than or equal to 300. But in a relational database, we can use a range of numbers to specify the same condition.

Instead of using the comparison operators greater than or equal to, we use the comparison operator 'between and.' 'Between and' compares two values.

The values in the range are inclusive. In this case,we rewrite the query to specify the WHERE clause as where pages between 290 and 300.

The result set is the same, but the SELECT statement is easier and quicker to write.

In some cases, there are data values that cannot be grouped under ranges.

For example, if we want to know which countries the authors are from.

If we wanted to retrieve authors from Australia or Brazil,

we could write the SELECT statement with the WHERE clause repeating the two country values.

However, what if we want to retrieve authors from Canada, India, and China?

The WHERE clause would become very long repeatedly listing the required country conditions.

Instead, we can use the IN operator. The IN operator allows us to specify a set of values in a WHERE clause. This operator takes a list of expressions to compare against.

In this case the countries Australia or Brazil. Now you can describe how to simplify a SELECT statement by using string patterns, ranges, or sets of values.

Using String Patterns, Ranges

Using String Patterns, Ranges, and Sets

- 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

Retrieving rows from a table

```
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	978-0-98006283-1-1	300	DB-A02	Teaches you the fundamentals
B2	Database Fundamentals	1	2010	24.99	978-0-98466283-5-1	280	DB-A01	Teaches you the essentials of
B3	Getting started with DB2 App Dev	1	2011	35.99	978-0-98086283-4-1	345	DB-A03	Teaches you the essentials of
B4	Getting started with WAS CE	1	2010	49.99	978-0-98946283-3-1	458	DB-A04	Teaches you the essentials of

```
4 record(s) selected.
```

```
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.
```

Retrieving rows - using a String Pattern

- WHERE requires a predicate
- A predicate is an expression that evaluates to True, False, or Unknown
- Use the LIKE predicate with string patterns for the search

Example:

- WHERE <columnname> LIKE <string pattern>

```
WHERE firstname LIKE R%
```

Retrieving rows - using a String Pattern

```
db2 => select firstname from Author
         WHERE firstname like 'R%'

Firstname
Raul
Rav
     2 record(s) selected.
```

Retrieving rows - using a Range

```
db2 => select title, pages from Book
          WHERE pages >= 290 AND pages <= 300
```

Title	Pages
Database Fundamentals	300
Getting started with DB2 App Dev	298

2 record(s) selected.

```
db2 => select title, pages from Book
          WHERE pages between 290 and 300
```

Title	Pages
Database Fundamentals	300
Getting started with DB2 App Dev	298

2 record(s) selected.

Retrieving rows - using a Set of Values

```
db2 => select firstname, lastname, country
from Author
          WHERE country='AU' OR country='BR'
```

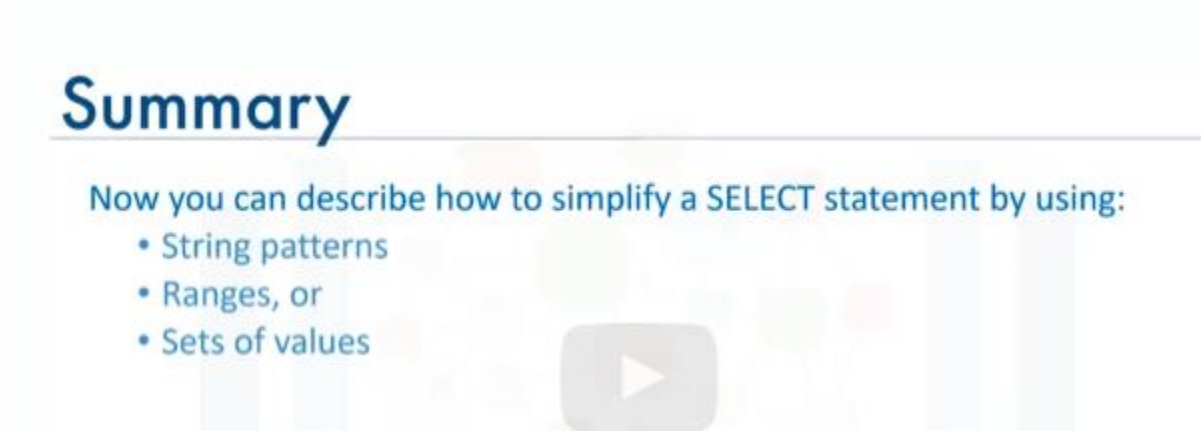
Firstname	Lastname	Country
Xiqiang	Ji	AU
Juliano	Martins	BR

2 record(s) selected.

```
db2 => select firstname, lastname, country
from Author
          WHERE country IN ('AU','BR')
```

Firstname	Lastname	Country
Xiqiang	Ji	AU
Juliano	Martins	BR

2 record(s) selected.



Sorting Result Sets

Hello, and welcome to sorting SELECT statement results sets.

In this video, we will learn about some advanced techniques in retrieving data from

a relational database table and sorting how the result set displays.

At the end of this lesson, you will be able to describe how to sort the result set by either ascending or descending order and 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.

In 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. To display the result set in alphabetical order, we add the order by clause to the select statement.

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. To sort in descending order, use the key word" desc." The result set is now sorted according to the column specified,

which is title, and is sorted in descending order. Notice the order of the first three rows.

The first three words of the title are the same, so the sorting starts from the point where the characters differ.

Another way of specifying the sort column is to indicate the column sequence number.

In this example, select title, pages from book order by two, indicates the column sequence number in the query for the sorting order. Instead of specifying the column name pages,

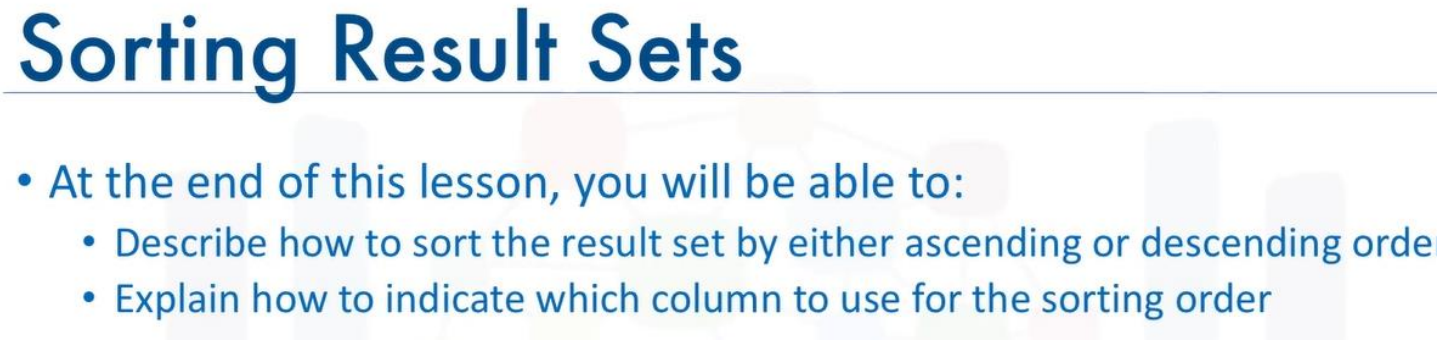
the number two is used. In the select statement, the second column specified in the column list is pages, so the sort order is based on the values in the pages column.

In this case, the pages column indicates the number of pages in the book.

As you can see, the result set is in ascending order by number of pages.

Now you can describe how to sort the result set by either ascending or

descending order, and explain how to indicate which column to use for the sorting order.



Sorting the Result Set

```
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	978-0-98006283-1-1	300	DB-A02	Teaches you the fundamentals
B2	Database Fundamentals	1	2010	24.99	978-0-98666283-5-1	280	DB-A01	Teaches you the essentials of
B3	Getting started with DB2 App Dev	1	2011	35.99	978-0-98086283-4-1	345	DB-A03	Teaches you the essentials of
B4	Getting started with WAS CE	1	2010	49.99	978-0-98946283-3-1	458	DB-A04	Teaches you the essentials of

```
4 record(s) selected.
```

```
db2 => select title from Book

Title
----------------------------------------
Getting started with DB2 Express-C
Database Fundamentals
Getting started with DB2 App Dev
Getting started with WAS CE

  4 record(s) selected.
```

Using the ORDER BY clause

```
db2 => select title from Book

Title
----------------------------------------
Getting started with DB2 Express-C
Database Fundamentals
Getting started with DB2 App Dev
Getting started with WAS CE

  4 record(s) selected.
```

```
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.
```

By default the result set is sorted in ascending order

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

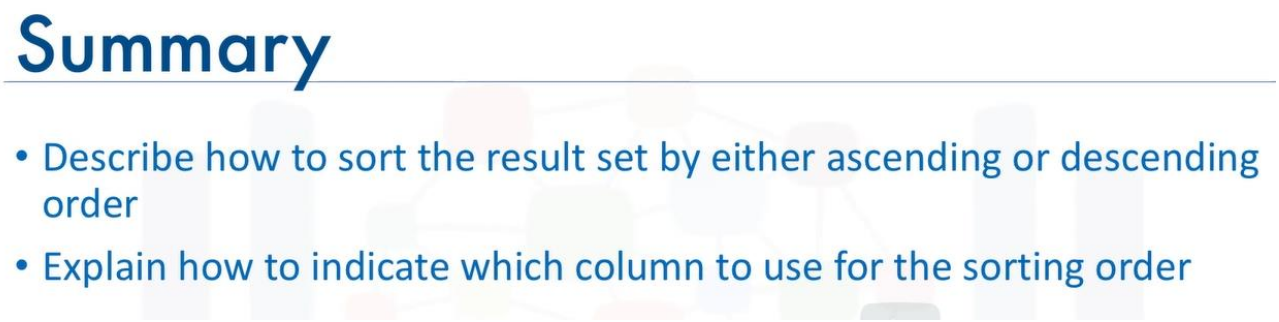
```
db2 => select title from Book
          ORDER BY title DESC

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

Hello and welcome to Grouping Select Statement Result Sets. In this video, we will learn about some advanced techniques in retrieving data from a relational database table, and sorting, and grouping how the results set displays.

At the end of this lesson, you will be able to explain how to eliminate duplicates from a result set and 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.

Notice the column heading for the second column and the result set.

The numeric value "2" displays as a column name because the column name is not directly available in the table. The second column in the result set was calculated by the count function.

Instead of using the column named "2," we can assign a column name to the result set.

We do this using the "as" keyword. In this example, we change the derived column name

"2" to column name "Count" using the "as count" keyword. This helps clarify the meaning of the result set. Now that we have the count of authors from different countries, we can further restrict the number of rows by passing some conditions.

For example, we can check if there are more than four authors from the same country.

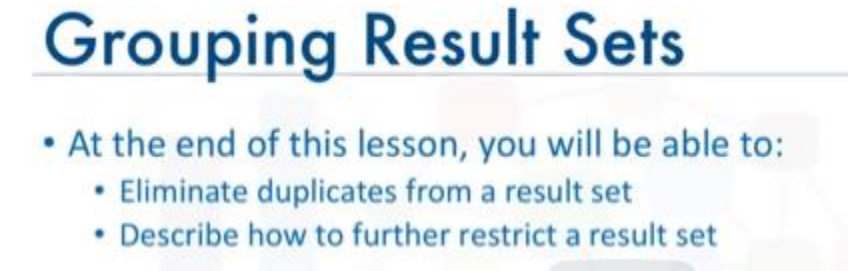
To set a condition to a "group by" clause, we use the keyword "having". The "having" clause is used in combination with the "group by" clause.

It is very important to note that the "where" clause is for the entire result set, but the "having" clause works only with the "group by" clause. To check if there are more than four authors from the same country, we add the following to the select statement, having count country greater than four.

Only countries that have five or more authors from that country are listed in the result set.

In this example, those countries are China with six authors and India, also with six authors.

Now you can explain how to eliminate duplicates from a result set and describe how to further restrict a result set.



Eliminating Duplicates - DISTINCT clause

```
db2 => select country from Author
          ORDER BY 1

Country
-------------------------------------------------
AU
BR
...
CN
CN
...
IN
IN
IN
...
RO
RO

      20 record(s) selected.
```

```
db2 => select distinct(country)
          from Author

Country
-------------------------------------------------
AU
BR
CA
CN
IN
RO

      6 record(s) selected.
```

GROUP BY clause

```
db2 => select country from Author
          ORDER BY 1

Country
-------------------------------------------------
AU
BR
...
CN
CN
...
IN
IN
IN
...
RO
RO

      20 record(s) selected.
```

```
db2 => select country, count(country)
          from Author GROUP BY country

Country              2
-------------------------------------------------
AU                   1
BR                   1
CA                   3
CN                   6
IN                   6
RO                   3

      6 record(s) selected.
```

GROUP BY clause

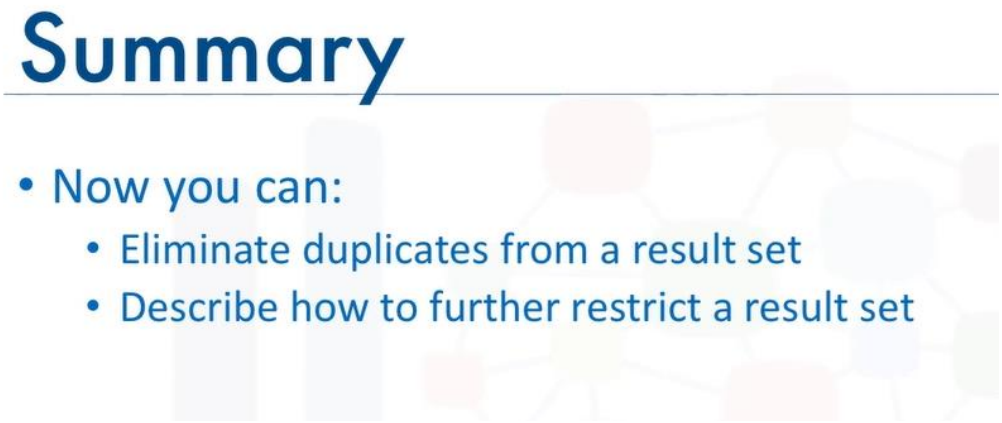
```
db2 => select country from Author
          ORDER BY 1

Country
-------------------------------------------------------
AU
BR
...
CN
CN
...
IN
IN
IN
...
RO
RO

      20 record(s) selected.
```

```
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.
```



Hands-on LAB: String Patterns, Sorting & Grouping (1 Hr)

Effort: 1 hour

Objective(s)

At the end of this lab you will be able to:

- Work with string patterns in data from database
- Sort and group the data in the database

LAB: String Patterns, Sorting & Grouping

To complete the Lab for this week, please refer to the instructions in the Lab instructions document.

Lab instructions:

LAB: String Patterns, Sorting & Grouping

Effort: 30 mins

The practice problems for this Lab will provide hands on experience with string patterns, sorting result sets and grouping result sets. You will also learn how to run SQL scripts to create several tables at once, as well as how to load data into tables from .csv files.

HR Database

We will be working on a sample HR database for this Lab. This HR database schema consists of 5 tables called EMPLOYEES, JOB_HISTORY, JOBS, DEPARTMENTS and LOCATIONS. Each table has a few rows of sample data The following diagram shows the tables for the HR database.

SAMPLE HR DATABASE TABLES

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

JOB_HISTORY

EMPL_ID	START_DATE	JOBS_ID	DEPT_ID
E1001	2000-01-30	100	2
E1002	2010-08-16	200	5
E1003	2016-08-10	300	5

JOBS

JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY
100	Sr. Architect	60000	100000
200	Sr.SoftwareDeveloper	60000	80000
300	Jr.SoftwareDeveloper	40000	60000

DEPARTMENTS

DEPT_ID_DEP	DEP_NAME	MANAGER_ID	LOC_ID
2	Architect Group	30001	L0001
5	Software Development	30002	L0002
7	Design Team	30003	L0003
5	Software	30004	L0004

LOCATIONS

LOCT_ID	DEP_ID_LOC
L0001	2
L0002	5
L0003	7

To complete this lab you will utilize Db2 database service on IBM Cloud as you did for the previous lab. There are three parts to this lab:

I. Creating tables

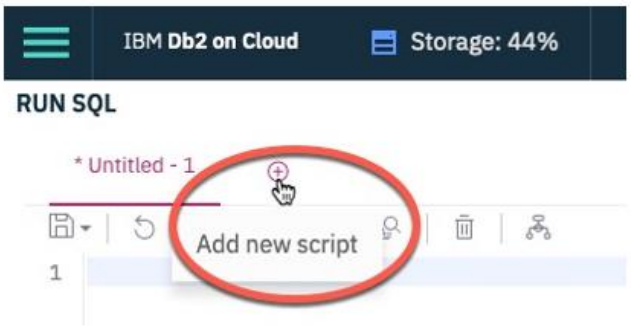
II. Loading data into tables

III.Composing and running queries

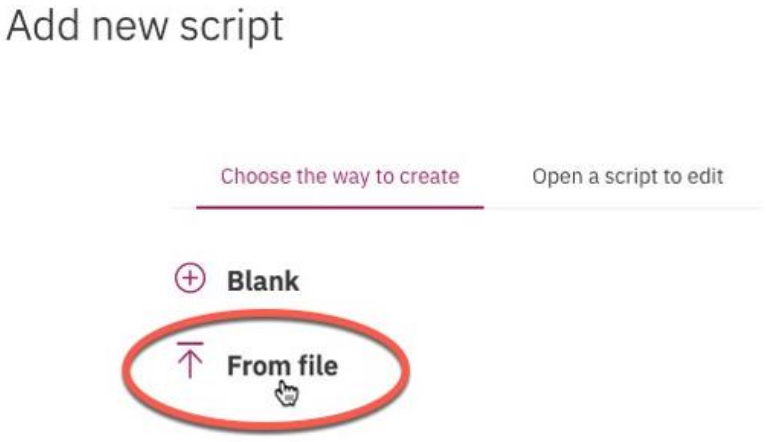
If you do not yet have access to Db2 on IBM Cloud, please refer to Lab Instructions in the Module/Week 1.

Rather than create each table manually by typing the DDL commands in the SQL editor, you will execute a script containing the create table commands for all the tables. Following step by step instructions are provided to perform this:

1. Download the script file **Script_Create_Tables.sql** provided on the Lab page
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

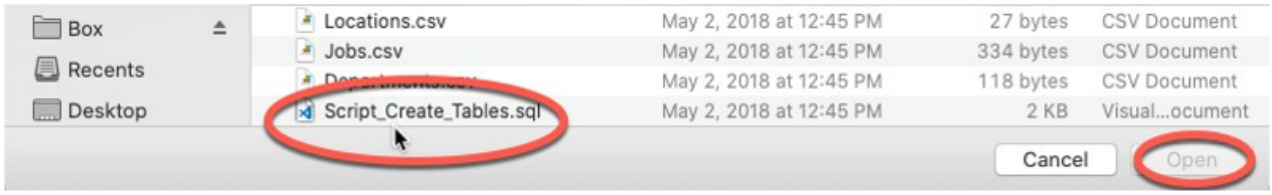


Click on `From File`

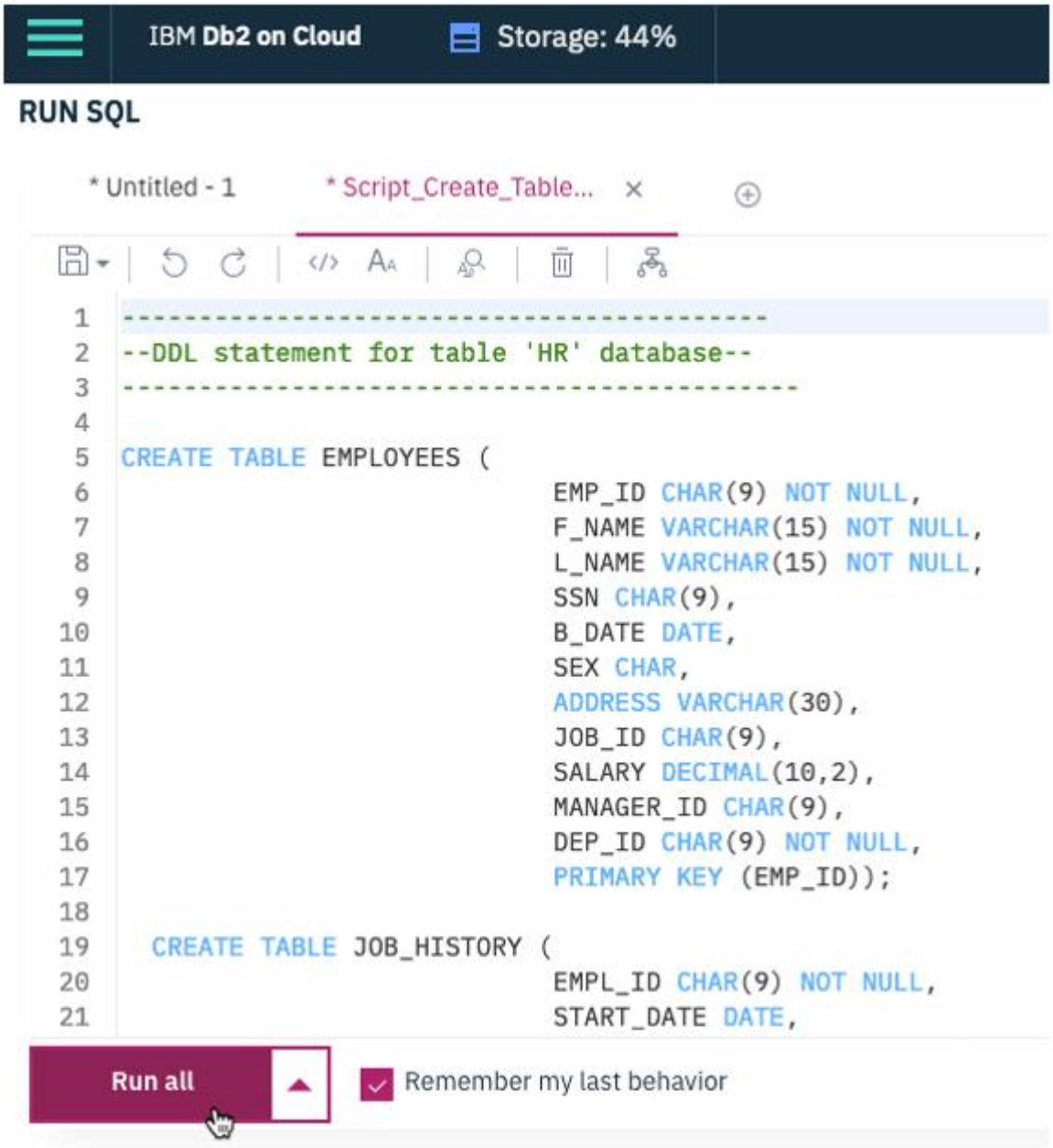


Locate the file Script_Create_Tables.sql that you downloaded to your computer earlier and open it.

Locate the file Script_Create_Tables.sql that you downloaded to your computer earlier and open it.

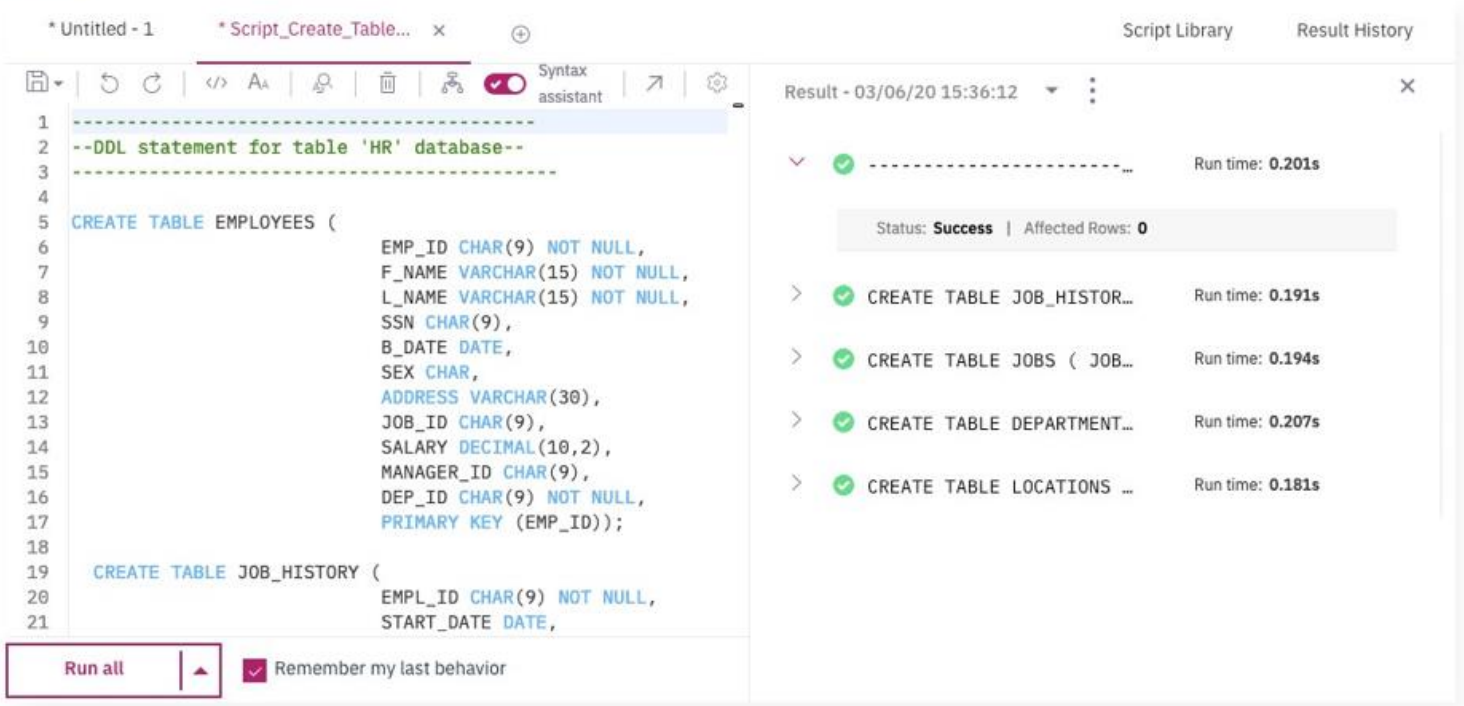


4. Once the statements are in the SQL Editor tool , you can run the queries against the database by selecting the `Run All` button.

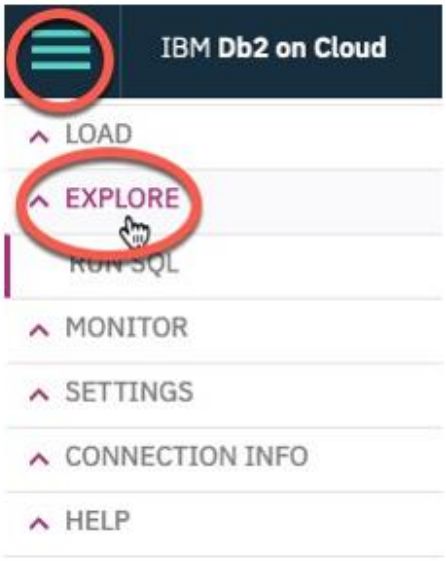


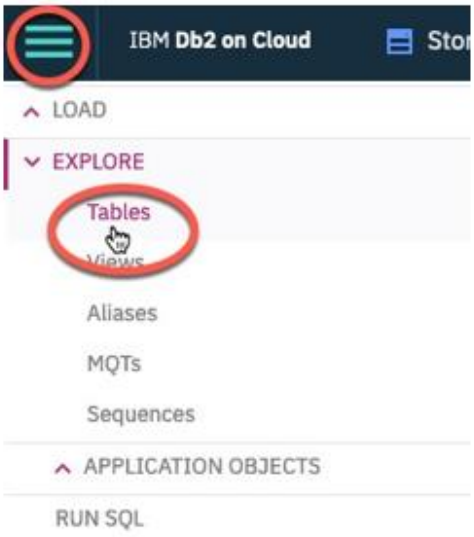
5. On the right side of the SQL editor window you will see a Result section. Clicking on a query in the Result scetion will show the execution details of the job - whether it ran successfully, or had any errors or warnings. Ensure your queries ran successfully and created all the tables.

RUN SQL

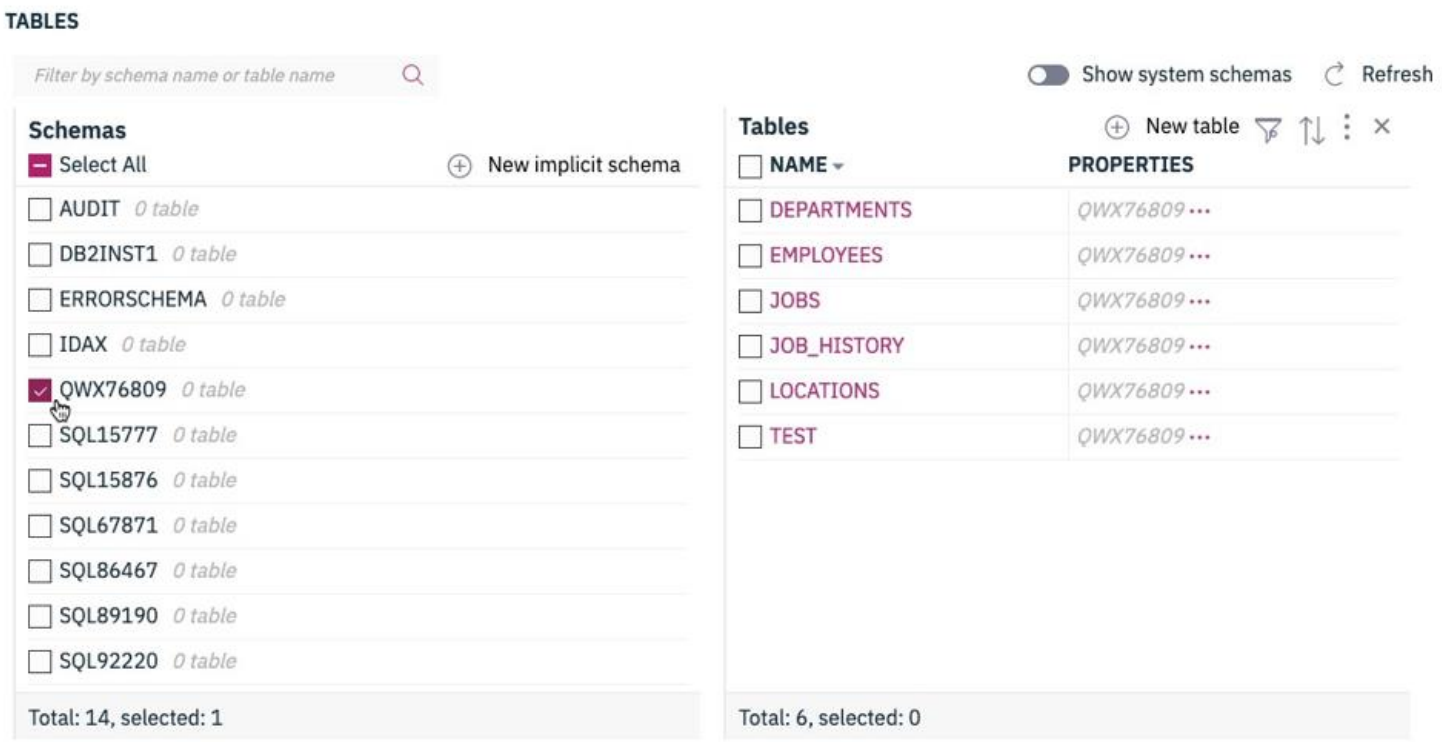


6. Now you can look at the tables you created. Navigate to the three bar menu icon, select Explore, then click on Tables

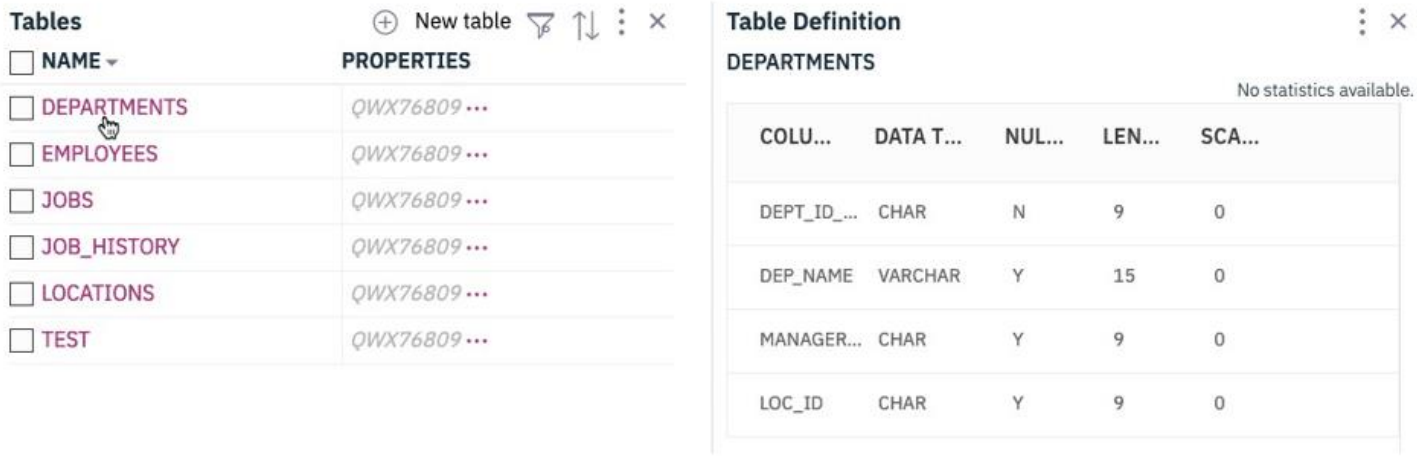




Select the Schema corresponding to your Db2 userid. It is typically starts with 3 letters (not SQL) followed by 5 numbers (but will be different from the QWX76809 example below). Then on the right side of the screen you should see the 5 newly created tables listed – DEPARTMENTS, EMPLOYEES, JOBS, JOB_HISTORY, and LOCATIONS (plus any other tables you may have created in previous labs e.g. INSTRUCTOR, TEST, etc.).



Click on any of the tables and you will see its SCHEMA definition (that is list of columns, their data types, etc).



Part II: LOADING DATA

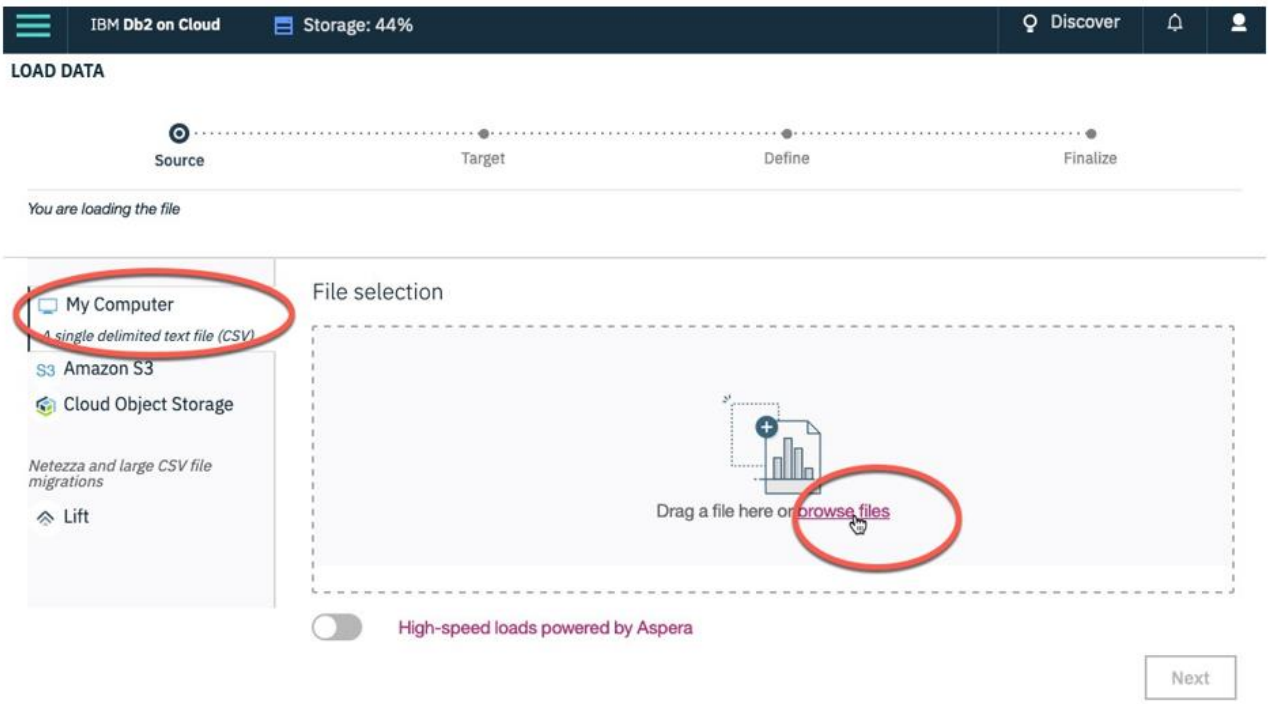
Now let us see how data can be loaded into Db2. We could manually insert each row into the table one by one but that would take a long time. Instead, Db2 (and almost every other database) allows you to Load data from .CSV files.

Please follow the steps below which explains the process of loading data into the tables we created earlier.

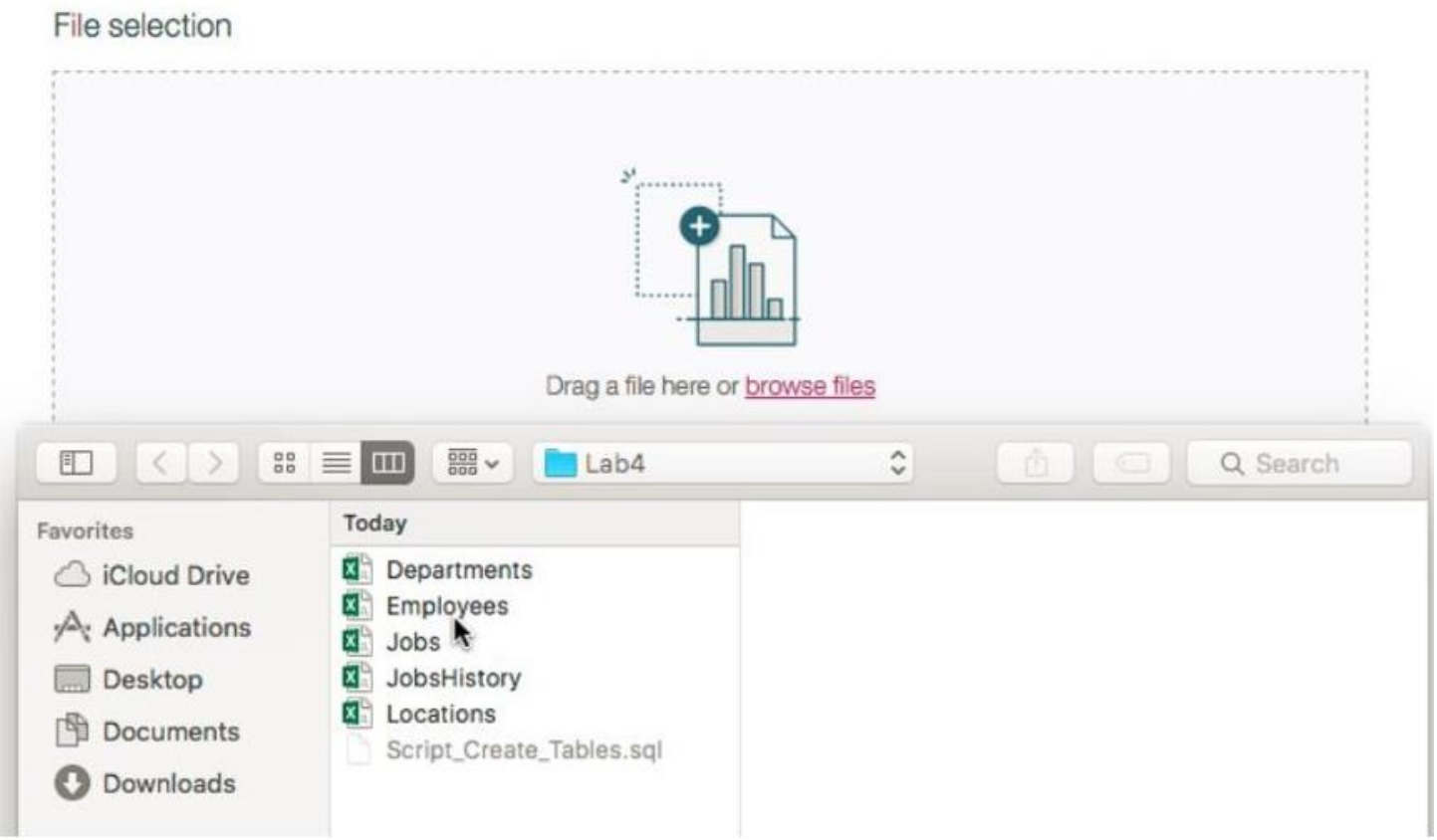
Download the 5 required data source files from the lab page in the course: (Employees.csv,Departments.csv,Jobs.csv,JobsHistory.csv,Locations.csv) to your computer:

First let us learn how to load data into the Employees table that we created earlier. From the 3 bar menu icon, select Load then Load Data:

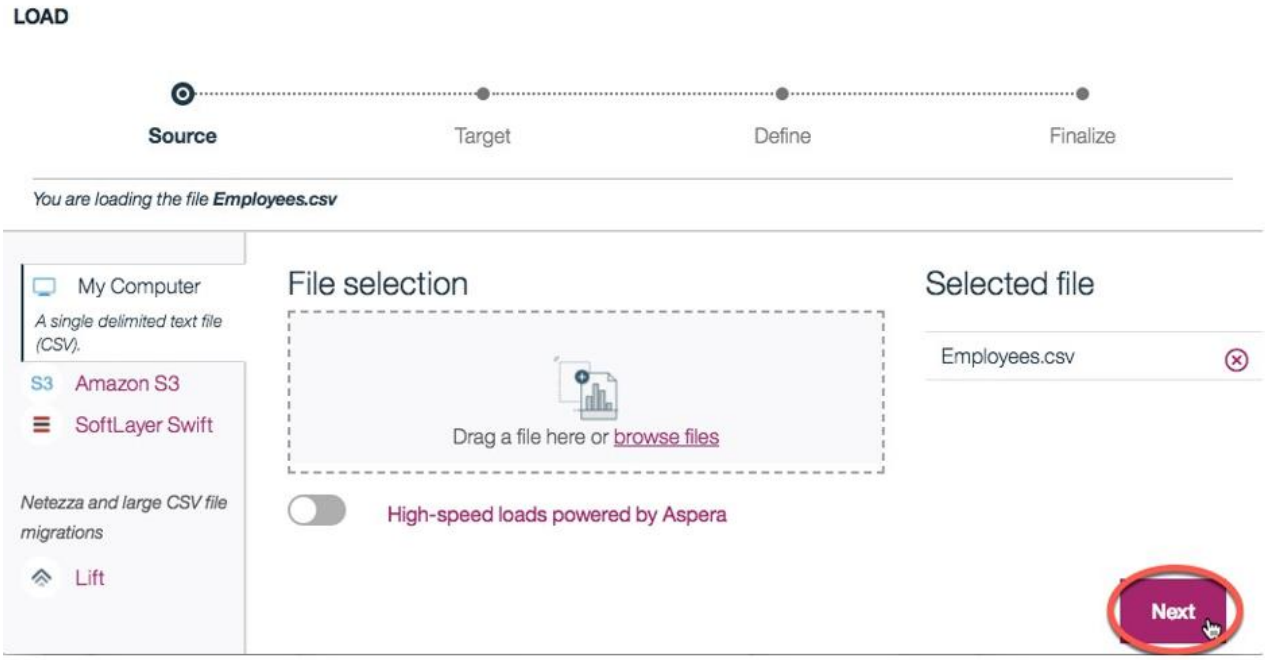
On the Load page that opens ensure My Computer is selected as the source. Click on the browse files link.



3. Choose the file `Emloyees.csv` that you downloaded to your computer and click Open.



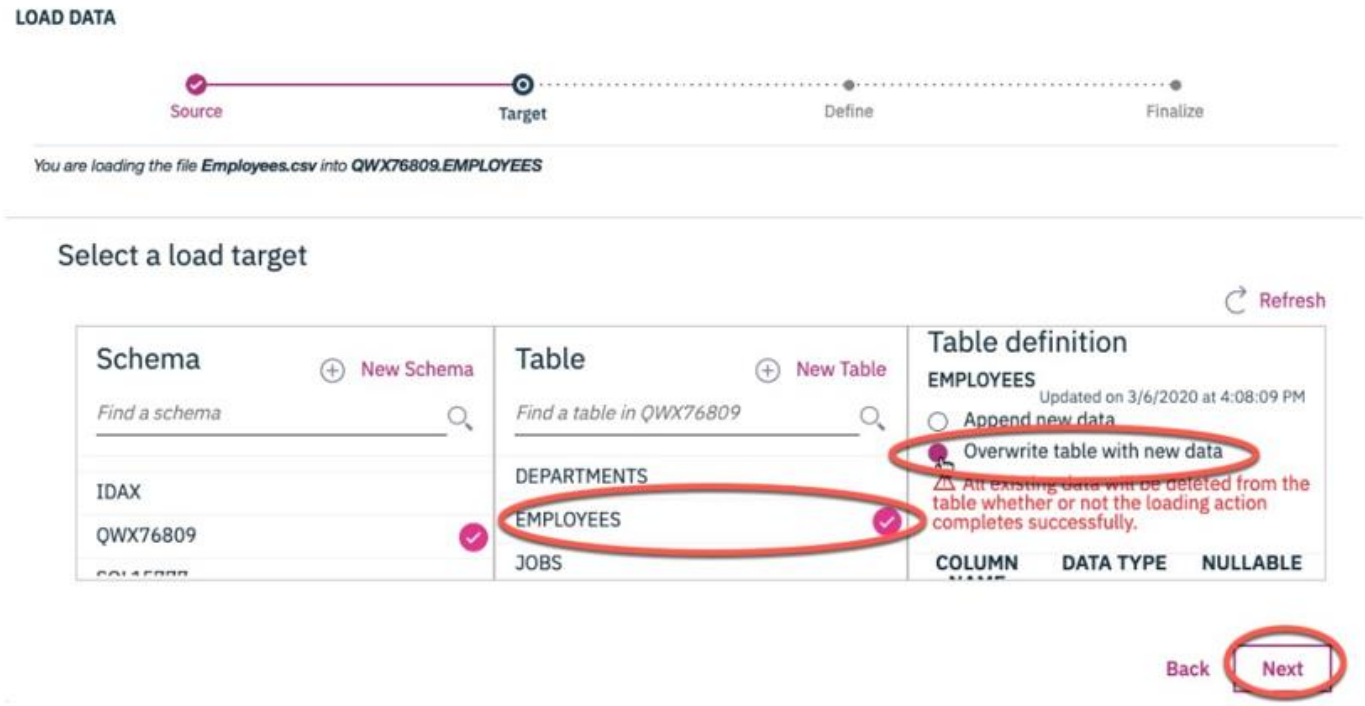
4. Once the File is selected click Next in the bottom right corner.



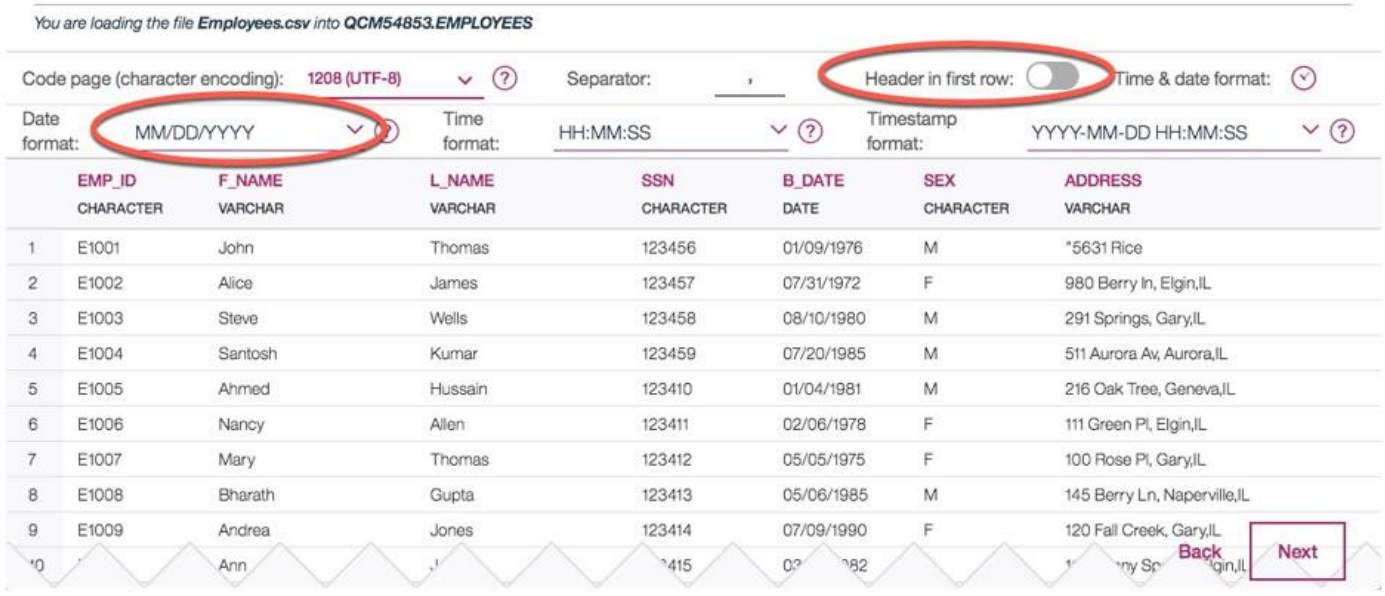
5. Select the schema for your Db2 Userid

NOTE: if you only see 2-3 schemas and not your Db2 schema then scroll down in that list till you see the desired one in which you previously created the tables.

It will show all the tables that have been created in this Schema previously, including the Employees table. Select the EMPLOYEES table, and choose Overwrite table with new data then click Next.

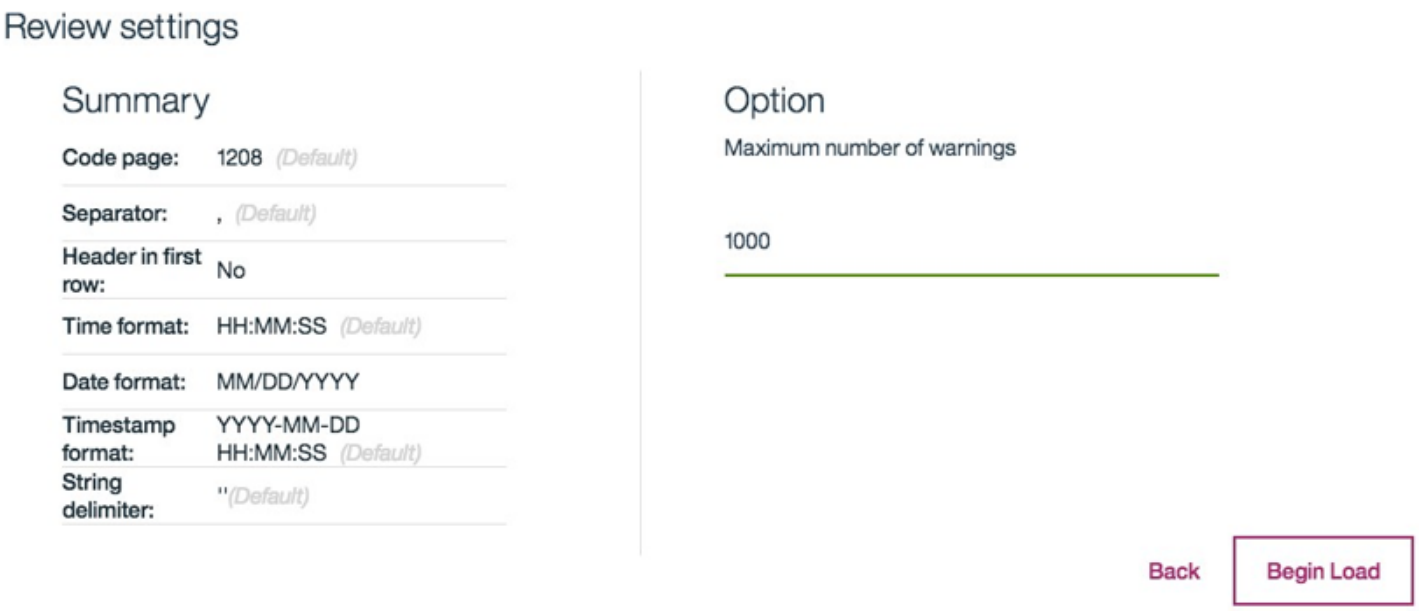


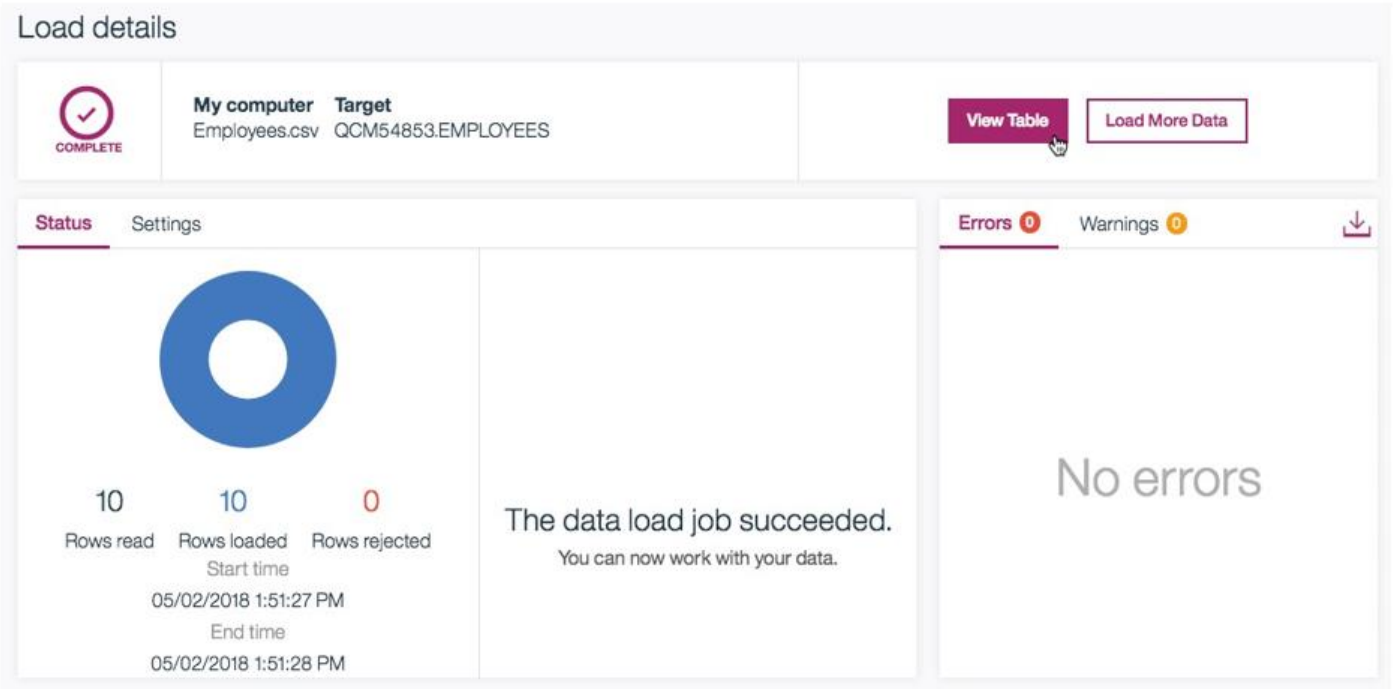
6. Since our source data files do not contain any rows with column labels, turn off the setting for Header in first row. Also, click on the down arrow next to Date format and choose `MM/DD/YYYY` since that is how the date is formatted in our source file.



7. Click `Next`. Review the Load setting and click `Begin Load` in the top-right corner

8. After Loading is complete you will notice that we were successful in loading all 10 rows of the Employees table. If there are any Errors or Warnings you can view them on this screen.





9. You can see the data that was loaded by clicking on the View Table. Alternatively you can go into the Explore page and page select the correct schema, then the EMPLOYEES table, and click View Data.

QCM54853.EMPLOYEES

Delete Table Export to CSV

	EMP_ID CHARACTER(9)	F_NAME VARCHAR(15)	L_NAME VARCHAR(15)	SSN CHARACTER(9)	B_DATE DATE	SEX CHARACTER(1)	ADDRESS VARCHAR(30)	JOB_ID CHARACTER(9)
1	E1001	John	Thomas	123456	1976-01-09	M	5631 Rice, OakPark,	100
2	E1002	Alice	James	123457	1972-07-31	F	980 Berry ln, Elgin,IL	200
3	E1003	Steve	Wells	123458	1980-08-10	M	291 Springs, Gary,IL	300
4	E1004	Santosh	Kumar	123459	1985-07-20	M	511 Aurora Av, Aurora	400
5	E1005	Ahmed	Hussain	123410	1981-01-04	M	216 Oak Tree, Geneva	500
6	E1006	Nancy	Allen	123411	1978-02-06	F	111 Green Pl, Elgin,IL	600
7	E1007	Mary	Thomas	123412	1975-05-05	F	100 Rose Pl, Gary,IL	650
8	E1008	Bharath	Gupta	123413	1985-05-06	M	145 Berry Ln, Naper	660
9	E1009	Andrea	Jones	123414	1990-07-09	F	120 Fall Creek, Gary,	234
10	E1010	Ann	Jacob	123415	1982-03-30	F	111 Britany Springs,E	220

10. Now its your turn to load the remaining 4 tables of the HR database – Locations, JobHistory, Jobs, and Departments. Please follow the steps above to load the data from the remaining source files.

Question 1: Were there any warnings loading data into the JOBS table? What can be done to resolve this?

Hint: View the data loaded into this table and pay close attention to the JOB_TITLE column.

Question 2: Did all rows from the source file load successfully in the DEPARTMENT table? If not, are you able to figure out why not?

Hint: Look at the warning. Also, note the Primary Key for this table

Part III: COMPOSING AND RUNNING QUERIES

You created the tables for the HR database schema and also learned how to load data into these tables. Now try and work on a few advanced DML queries that were introduced in this module.

Follow these steps to create and run the queries indicated below

Navigate to the Run SQL tool in Db2 on Cloud

Compose query and run it.

Check the Logs created under the Results section. Looking at the contents of the Log explains whether the SQL statement ran successfully. Also look at the query results to ensure the output is what you expected.

Query 1: Retrieve all employees whose address is in Elgin,IL

Hint: Use the LIKE operator to find similar strings

Query 2: Retrieve all employees who were born during the 1970's.

Hint: Use the LIKE operator to find similar strings

Query 3: Retrieve all employees in department 5 whose salary is between 60000 and 70000 .

Hint: Use the keyword BETWEEN for this query

Query 4A: Retrieve a list of employees ordered by department ID.

Hint: Use the ORDER BY clause for this query

Query 4B: Retrieve a list of employees ordered in descending order by department ID and within each department ordered alphabetically in descending order by last name.

Query 5A: For each department ID retrieve the number of employees in the department.

Hint: Use COUNT(*) to retrieve the total count of a column, and then GROUP BY

Query 5B: For each department retrieve the number of employees in the department, and the average employees salary in the department.

Hint: Use COUNT(*) to retrieve the total count of a column, and AVG() function to compute average salaries, and then group

Query 5C: Label the computed columns in the result set of Query 5B as NUM_EMPLOYEES and AVG_SALARY.

Hint: Use AS "LABEL_NAME" after the column name

Query 5D: In Query 5C order the result set by Average Salary.

Hint: Use ORDER BY after the GROUP BY

Query 5E: In Query 5D limit the result to departments with fewer than 4 employees.

Hint: Use HAVING after the GROUP BY, and use the count() function in the HAVING clause instead of the column label.

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

BONUS Query 6: Similar to 4B but instead of department ID use department name. Retrieve a list of employees ordered by department name, and within each department ordered alphabetically in descending order by last name.

Hint: Department name is in the DEPARTMENTS table. So your query will need to retrieve data from more than one table. Don't worry if you are not able to figure this one out … we'll cover working with multiple tables in the next lesson.

In this lab you learned how to work with string patterns, sorting result sets and grouping result sets.

Thank you for completing this lab! See solutions on the following page

Lab Solutions

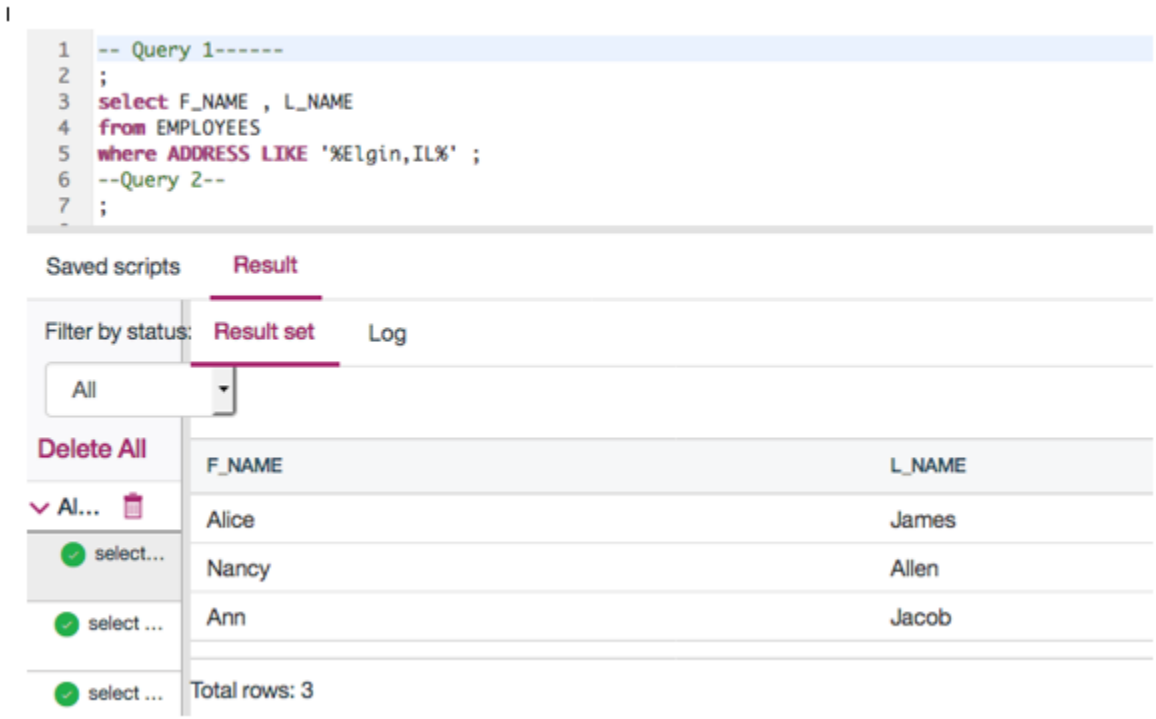
Please follow these steps to get the answers to the queries:

1. Navigate to the Run SQL page on Db2 on Cloud.

2. Download the script file(Module4_Queries.txt) or text files(Modules4_Queries.sql). Open the file with extension .sql in the editor

3. Run the queries. Looking at the contents of the Log explains that the SQL statement that we ran was successful. Here are the results for the queries:

Query 1: Output



Query 2: Output

RUN SQL

Run Script Edit Favorites New tab

```
6  --Query 2--
7  ;
8  select F_NAME , L_NAME
9  from EMPLOYEES
10 where B_DATE LIKE '197%' ;
11 ---Query3--
12 :
```

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

Query 3: Output

```
11 ---Query3--
12 ;
13 select *
14 from EMPLOYEES
15 where (SALARY BETWEEN 60000 and 70000)  and DEP_ID = 5 ;
16 --Query4--
17 ;
```

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

Query 4A: Output

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

ed scripts | Result

r by status: | Result set | Log

ll

elete All

(1)... select F_...

(1)... select F_...

(1)... select F_...

(1)... select F_...

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

Query 4B: Output

```
1  select F_NAME, L_NAME, DEP_ID
2  from EMPLOYEES
3  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

Query 5A: Output

```
select DEP_ID, COUNT(*)
from EMPLOYEES
group by DEP_ID;
```

ved scripts | Result

er by status: | Result set | Log

All

lete All

l... select...

l... select ...

DEP_ID	2
2	3
5	4
7	3

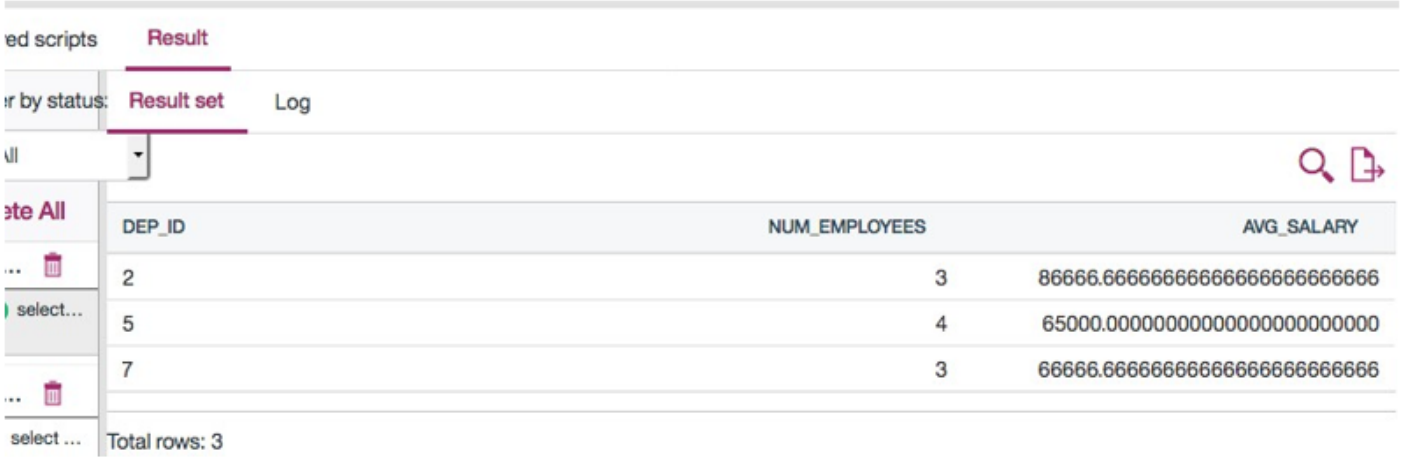
Total rows: 3

Query 5B: Output

```
1  select DEP_ID, COUNT(*), AVG(SALARY)
2  from EMPLOYEES
3  group by DEP_ID;
```

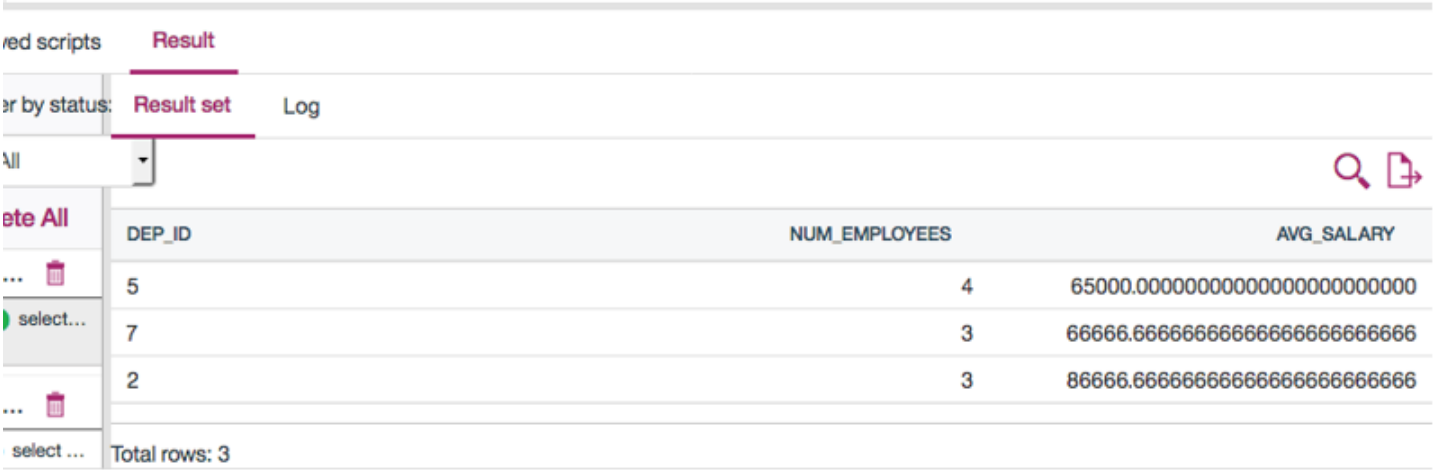
Query 5C: Output

```
select DEP_ID, COUNT(*) AS "NUM_EMPLOYEES", AVG(SALARY) AS "AVG_SALARY"
from EMPLOYEES
group by DEP_ID;
```



Query 5D: Output

```
select DEP_ID, COUNT(*) AS "NUM_EMPLOYEES", AVG(SALARY) AS "AVG_SALARY"
from EMPLOYEES
group by DEP_ID
order by AVG_SALARY;
```



ved scripts | Result

er by status: | Result set | Log

All

ete All

select...

select ...

DEP_ID	NUM_EMPLOYEES	AVG_SALARY
5	4	65000.00000000000000000000
7	3	66666.66666666666666666666
2	3	86666.66666666666666666666

Total rows: 3

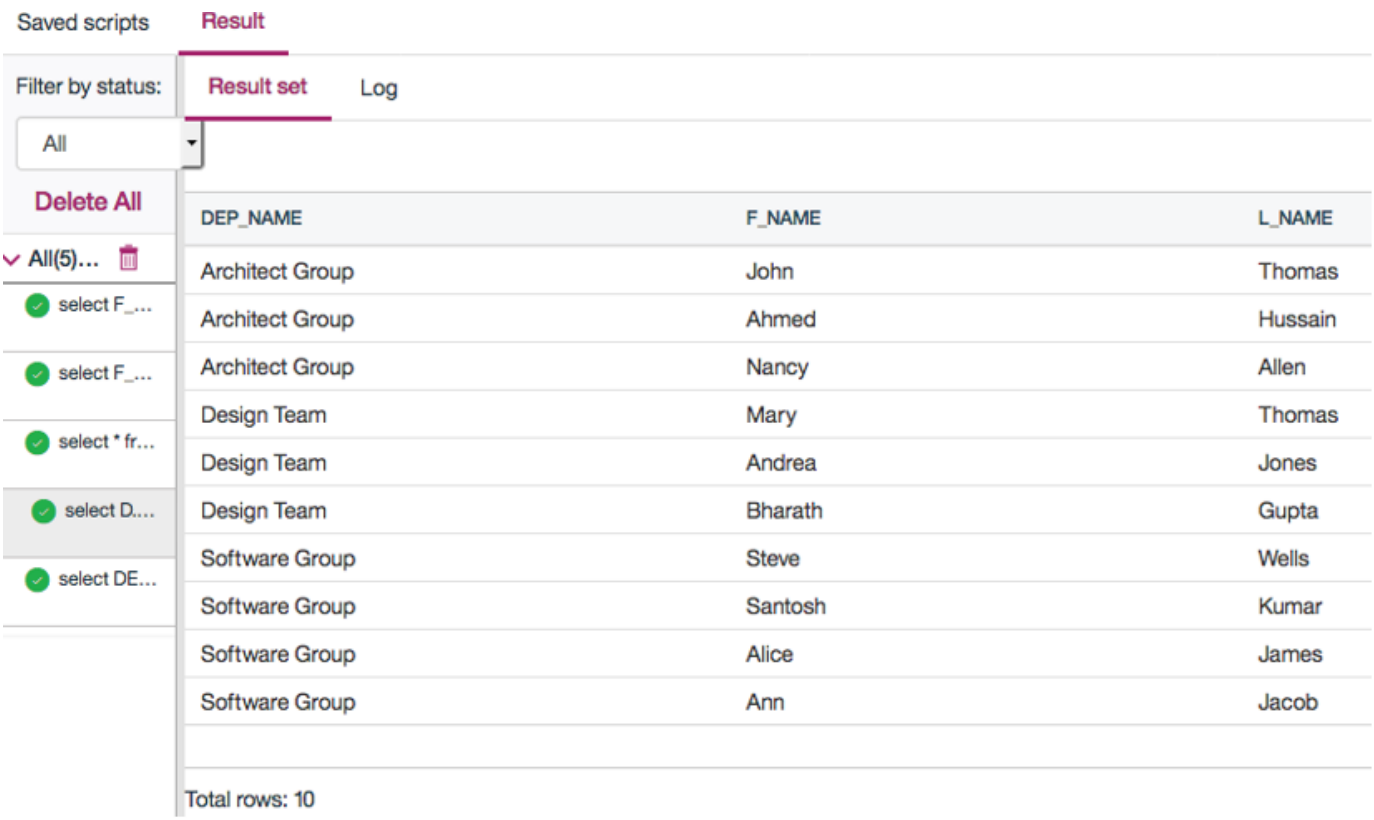
Query 5E: Output

```
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;
```

BONUS Query 6: Output

Note that in the Query below 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.

```
16  --Query4--
17  ;
18  select D.DEP_NAME , E.F_NAME, E.L_NAME
19  from EMPLOYEES as E, DEPARTMENTS as D
20  where E.DEP_ID = D.DEPT_ID_DEP
21  order by D.DEP_NAME, E.L_NAME desc ;
22  --Query5--
```


Saved scripts
Result
Filter by status:
All
Delete All
All(5)...
select F_...
select F_...
select * fr...
select D...
select DE...
Result set
Log
DEP_NAME
F_NAME
L_NAME
Architect Group
John
Thomas
Architect Group
Ahmed
Hussain
Architect Group
Nancy
Allen
Design Team
Mary
Thomas
Design Team
Andrea
Jones
Design Team
Bharath
Gupta
Software Group
Steve
Wells
Software Group
Santosh
Kumar
Software Group
Alice
James
Software Group
Ann
Jacob
Total rows: 10

Summary & Highlights

Bookmarked

Congratulations! At this point in the course, you know:

- You can use the WHERE clause to refine your query results.
- You can use the wildcard character (%) as a substitute for unknown characters in a pattern.
- You can use BETWEEN ... AND ... to specify a range of numbers.
- You can sort query results into ascending or descending order, using the ORDER BY clause to specify the column to sort on.
- You can group query results by using the GROUP BY clause.

Practice Quiz

Bookmark this page

Question 1

1 point possible (ungraded)

You want to select author's last name from a table, but you only remember the author's last name starts with the letter B, which string pattern can you use?

- ◯ SELECT lastname from author where lastname like 'B$'
- ◯ SELECT lastname from author where lastname like 'B%'
- ◯ SELECT lastname from author where lastname like 'B#'
- ◯ None of the above

Submit

Question 2

1 point possible (ungraded)

In a SELECT statement, which SQL clause controls how the result set is displayed?

- ORDER WITH clause
- ORDER IN clause
- ORDER BY clause

Submit

Question 3

1 point possible (ungraded)

Which of the following can be used in a SELECT statement to restrict a result set?

- DISTINCT
- WHERE
- HAVING
- All of the above

Submit

Question 4

1 point possible (ungraded)

When querying a table called Author that contains a list of authors and their country of residence, which of the following queries will return the number of authors from each country?

- ◯ SELECT Country, distinct(Country) FROM Author GROUP BY Country
- ◯ SELECT Distinct(Country) FROM Author
- ◯ SELECT Country, count(Country) FROM Author
- ◯ SELECT Country, count(Country) FROM Author GROUP BY Country

Submit

Question 5

1 point possible (ungraded)

You want to retrieve a list of books that have between 450 and 600 pages. Which clause would you add to the following SQL statement: SELECT Title, Pages FROM Book _____

- ◯ WHERE Pages = 450
- ◯ WHERE Pages 450 – 600
- ◯ IF Pages > = 450 and Pages < = 600
- ◯ WHERE Pages > = 450 and pages < = 600

Submit

Module 3 Graded Quiz

Bookmark this page

Graded Quiz due Dec 12, 2021 23:36 CST

Question 1

1 point possible (graded)

You want to select author's last name from a table, but you only remember the author's last name starts with the letter B, which string pattern can you use?

- SELECT lastname from author where lastname like 'B$'
- SELECT lastname from author where lastname like 'B%'
- SELECT lastname from author where lastname like 'B#'
- None of the above

Submit You have used 0 of 2 attempts Save

Question 2

1 point possible (graded)

Which of the following can be used in a SELECT statement to restrict a result set?

- All of the above
- WHERE
- DISTINCT
- HAVING

Submit You have used 0 of 2 attempts Save

Question 3

1 point possible (graded)

In a SELECT statement, which SQL clause controls how the result set is displayed?

- ◯ ORDER WITH clause
- ◯ ORDER IN clause
- ◯ ORDER BY clause

Submit You have used 0 of 2 attempts Save

Built-in Database Functions

Hello and welcome. In this Video, we'll go over SQL functions built into the database.

So let's get started. While it is very much possible to first fetch data from a database and

then perform operations on it from your applications and notebooks, most databases come with Built-in Functions.

These functions can be included in SQL statements, allowing you to perform operations on

data right within the database itself. Using database functions can significantly reduce

the amount of data that needs to be retrieved from the database. That is, reduces network traffic and use of bandwidth. When working with large data sets, it may be faster to use built in functions, rather than first retrieving the data into your application and then executing functions on the retrieved data.

Note that it is also possible to create your own functions, that is User-Defined Functions in the database; but that is a more advanced topic.

For the examples in this lesson, let's consider this PETRESCUE table in a database for a pet

rescue organization. It records rescue transaction details and includes the columns: ID,

animal, quantity, cost, and rescue date.

For the purposes of this lesson, we have populated it with several rows of data, as shown here.

What are aggregate or column functions? An aggregate function takes a collection of like values,

such as all of the values in a column, as input, and returns a single value or null.

Examples of aggregate functions include: sum, minimum, maximum, average, etc. Let's look at some examples based on the PETRESCUE table.

The SUM function is used to add up all the values in a column. To use the function, you write the column name within parenthesis, after the function name.

For example, to add up all the values in the COST column, select SUM (COST) from PETRESCUE.

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

It is possible to explicitly name the resulting column. For example, let's say we want to call the output column in the previous query, as SUM_OF_COST. select SUM(COST) as SUM_OF_COST from PETRESCUE.

Note the use of 'as' in this example. MINimum, as the name implies, is used to get the lowest value. Similarly, MAXimum is used to get the highest value.

For example, to get the maximum quantity of any animal rescue in a single transaction, select MAX(QUANTITY) from PETRESCUE.

Aggregate functions can also be applied on a subset of data instead of an entire column.

For example, to get the minimum quantity of the ID column for dogs.

select MIN(ID) from PETRESCUE where animal equals dog. The Average (AVG) function is used to return the average or the mean value. For example, to specify the average value of cost,

as: select AVG(COST) from PETRESCUE.

Note that we can perform mathematical operations between columns, and then apply aggregate functions on them. For example, to calculate the average cost per dog:

select AVG(COST divided by QUANTITY) from PETRESCUE where ANIMAL equals

Dog. In this case, the cost is for multiple units; so we first divide the cost by the quantity of the rescue.

Now let's look at the Scalar and String functions. Scalar functions perform operations on individual values. For example, to round up or down every value in the cost column

to the nearest integer, select ROUND (COST) from PETRESCUE. There is a class of scalar functions called string functions, that can be used for operations on strings.

That is char and varchar values. For example, to retrieve the length of each value in animal column, select LENGTH (ANIMAL) from PETRESCUE.

Uppercase and lowercase functions can be used to return uppercase or lowercase values of strings. For example, to retrieve animal values in uppercase: select UPPERCASE (ANIMAL) from PETRESCUE.

Scalar functions can be used in the where clause. For example, to get lowercase values of the animal column for cat, select star from PETRESCUE where LOWERCASE(ANIMAL) equals cat.

This type of statement is useful for matching values in the where clause, if you're not sure whether the values are stored in upper, lower or mixed case in the table.

You can also have one function operate on the output of another function. For example, to get unique cases for the animal column in uppercase: select DISTINCT (UPPERCASE(ANIMAL)) from PETRESCUE.

In this video, we looked at some built-in SQL aggregate functions, such as sum, minimum, maximum, and average We also looked at scalar and string functions, such as round, lowercase, and uppercase.

Built-in 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
```

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

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
```

SCALAR and STRING FUNCTIONS

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

Example 7. Retrieve the length of each value in ANIMAL:

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

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
```

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

Now let's practice using sub-queries and working with multiple tables. Use the PETRESCUE-CREATE.sql file provided to create the table and execute the queries in the last two videos.

ID	ANIMAL	QUANTITY	COST	RESCUEDATE
1	Cat	9	450.09	5/29/2018
2	Dog	3	666.66	6/1/2018
3	Dog	1	100	6/4/2018
4	Parrot	2	50	6/4/2018
5	Dog	1	75.75	6/10/2018
6	Hamster	6	60.6	6/11/2018
7	Cat	1	44.44	6/11/2018
8	Goldfish	24	48.48	6/14/2018
9	Dog	2	222.22	6/15/2018

Objectives

After completing this lab, you will be able to:

Compose and run sub-queries with multiple tables

Check query results and view log files

Compose and run the following queries. Check that the results are what you expect and remember to view the logs in the Results section for errors and warnings.

Note: The solutions are provided at the end of this lab, but please try to compose the queries on your own before checking the solutions.

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.

Exercise 2: Aggregate Functions

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

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.

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

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

Query A5: Enter a function that displays the average cost of rescuing a dog.

Exercise 3: Scalar and String Functions

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

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

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

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

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.

Exercise 4: Date and Time Functions

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

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

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

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

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

Lab Solutions

Exercise 2 Solutions: 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 Solutions: 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 Solutions: 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;
```

Date and Time Built-in Functions

we'll go over date and time SQL functions built into the database. So let's get started.

Most databases contain special data types for dates and times.

Db2 contains date, time, and timestamp types.

In Db2, Date has eight digits: for year, month, and day.

Time has six digits: hours, minutes, and seconds.

Timestamp has 20 digits: year, month, day, hour, minute, seconds,

and microseconds where double X represents month and six Zs or Zs represents

microseconds.

Functions exist to extract the day, month, day of month, day of week, day of year,

week, hour, minute, and second.

Let us look at some examples of queries for date and time functions.

The day function can be used to extract the day portion from a date.

For example, to get the day portion for each rescue date involving cat,

Select DAY (RESCUEDATE) from PETRESCUE where ANIMAL equals cat.

Date and time functions can be used in the where clause.

For example, to get the number of rescues during the month of May,

that is, from Month 5, select COUNT star from PETRESCUE

where MONTH (RESCUEDATE) equals 05. You can also perform date or time arithmetic.

For example, to find out what date it is three days after each rescue date,

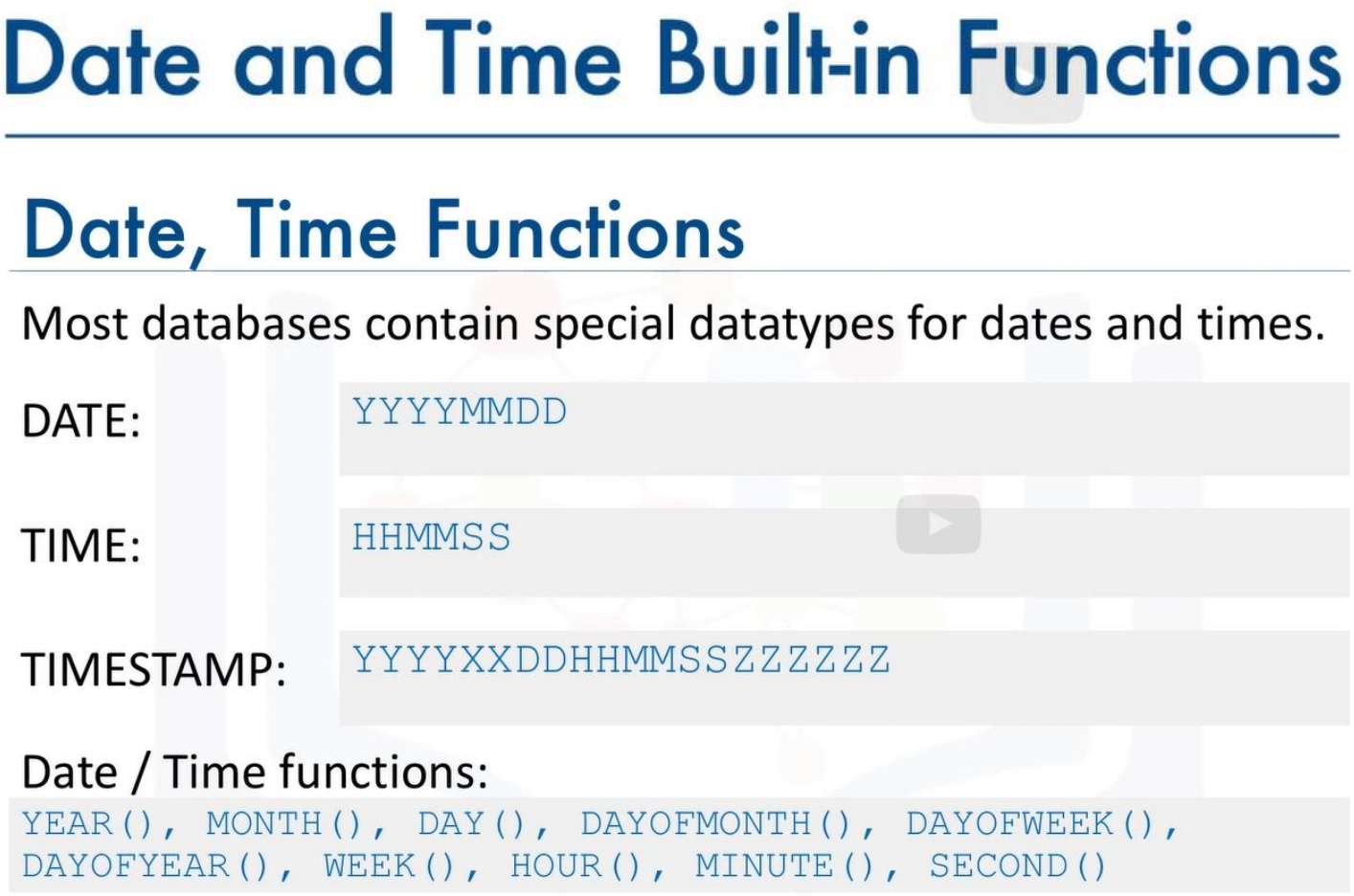
maybe you want to know this because the rescue needs to be processed within three days.

select (RESCUEDATE plus three DAYS) from PETRESCUE. Special registers current time

and current date are also available. For example, to find how many days have passed since each rescue date till now-

(CURRENT_DATE minus RESCUEDATE) from PETRESCUE. The result will be in years, months, days.

In this video, we looked at different types of built in SQL functions for working with dates and times.



Date, Time Functions (continued)

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
```

Date, Time Functions (continued)

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
```

Date or Time Arithmetic

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
```

Date or Time Arithmetic

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
```

Lab Solutions

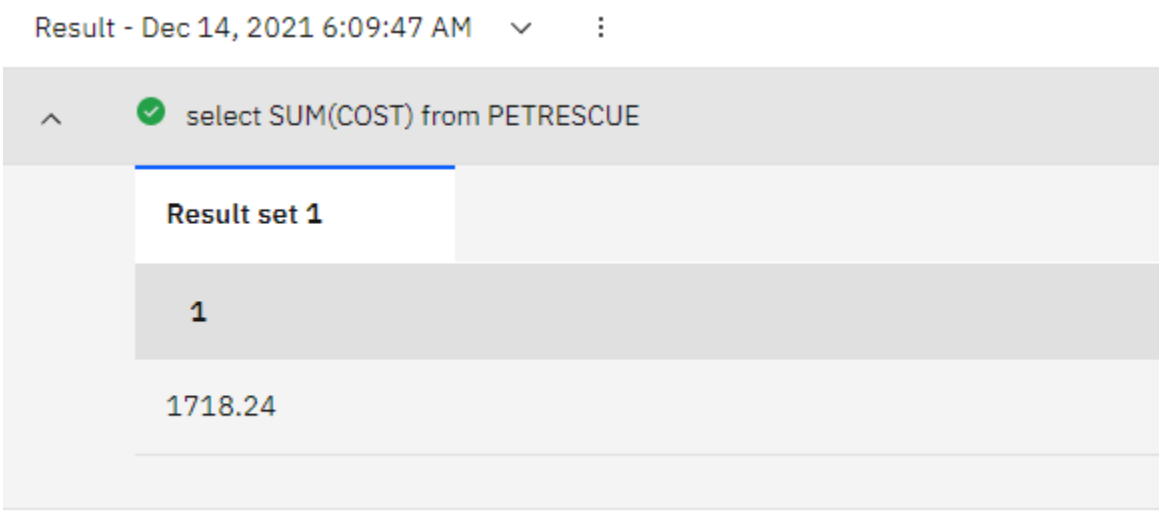
```
-- Drop the PETRESCUE table in case it exists
drop table PETRESCUE;
-- Create the PETRESCUE table
create table PETRESCUE (
    ID INTEGER NOT NULL,
    ANIMAL VARCHAR(20),
    QUANTITY INTEGER,
    COST DECIMAL(6,2),
    RESCUEDATE DATE,
    PRIMARY KEY (ID)
    );
-- Insert sample data into PETRESCUE table
insert into PETRESCUE values
    (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')
```

```
;
select SUM(COST) from PETRESCUE;
select SUM(COST) AS SUM_OF_COST from PETRESCUE;
select MAX(QUANTITY) from PETRESCUE;
select AVG(COST) from PETRESCUE;
select AVG(COST/QUANTITY) from PETRESCUE where ANIMAL = 'Dog';
select ROUND(COST) from PETRESCUE;
select LENGTH(ANIMAL) from PETRESCUE;
select UCASE(ANIMAL) from PETRESCUE;
select DISTINCT(UCASE(ANIMAL)) from PETRESCUE;
select * from PETRESCUE where LCASE(ANIMAL) = 'cat';
select DAY(RESCUEDATE) from PETRESCUE where ANIMAL = 'Cat';
select SUM(QUANTITY) from PETRESCUE where MONTH(RESCUEDATE)='05
select SUM(QUANTITY) from PETRESCUE where DAY(RESCUEDATE)='14';
select (RESCUEDATE + 3 DAYS) from PETRESCUE;
select (CURRENT DATE - RESCUEDATE) from PETRESCUE;
```

Exercise 2 Solutions: Aggregate Functions

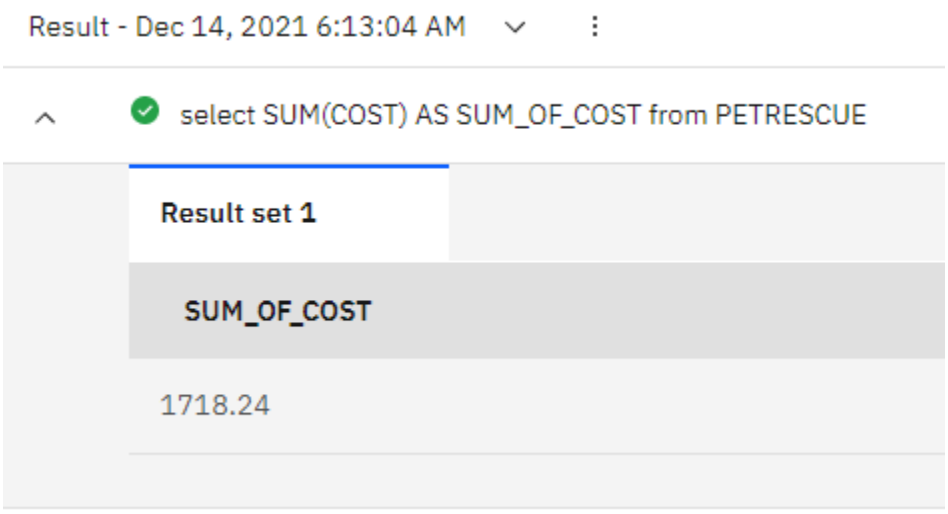
Query A1: Enter a function that calculates the total cost of all animal rescues in the PETRESCUE table.

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



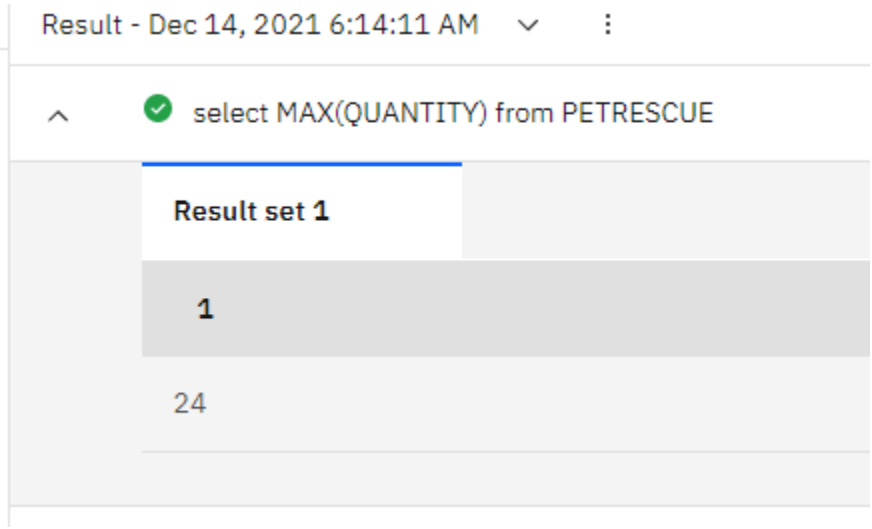
Query A2: Enter a function that displays the total cost of all animal rescues in the PETRESCUE table in a column called SUM_OF_COST.

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



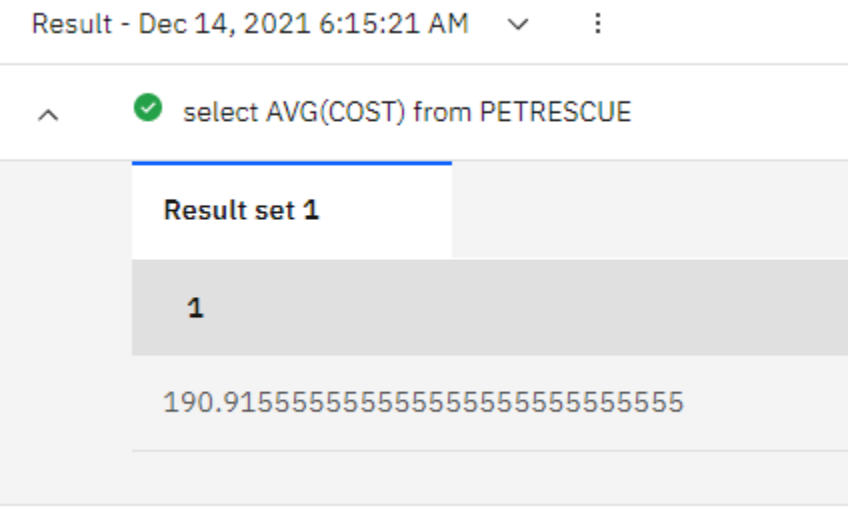
Query A3: Enter a function that displays the maximum quantity of animals rescued.

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



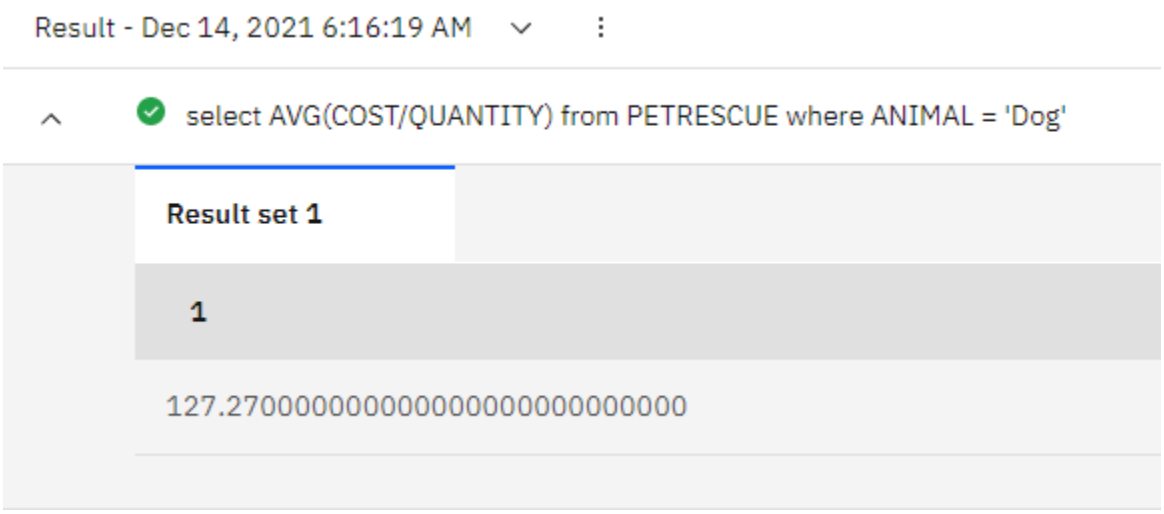
Query A4: Enter a function that displays the average cost of animals rescued.

```
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.*

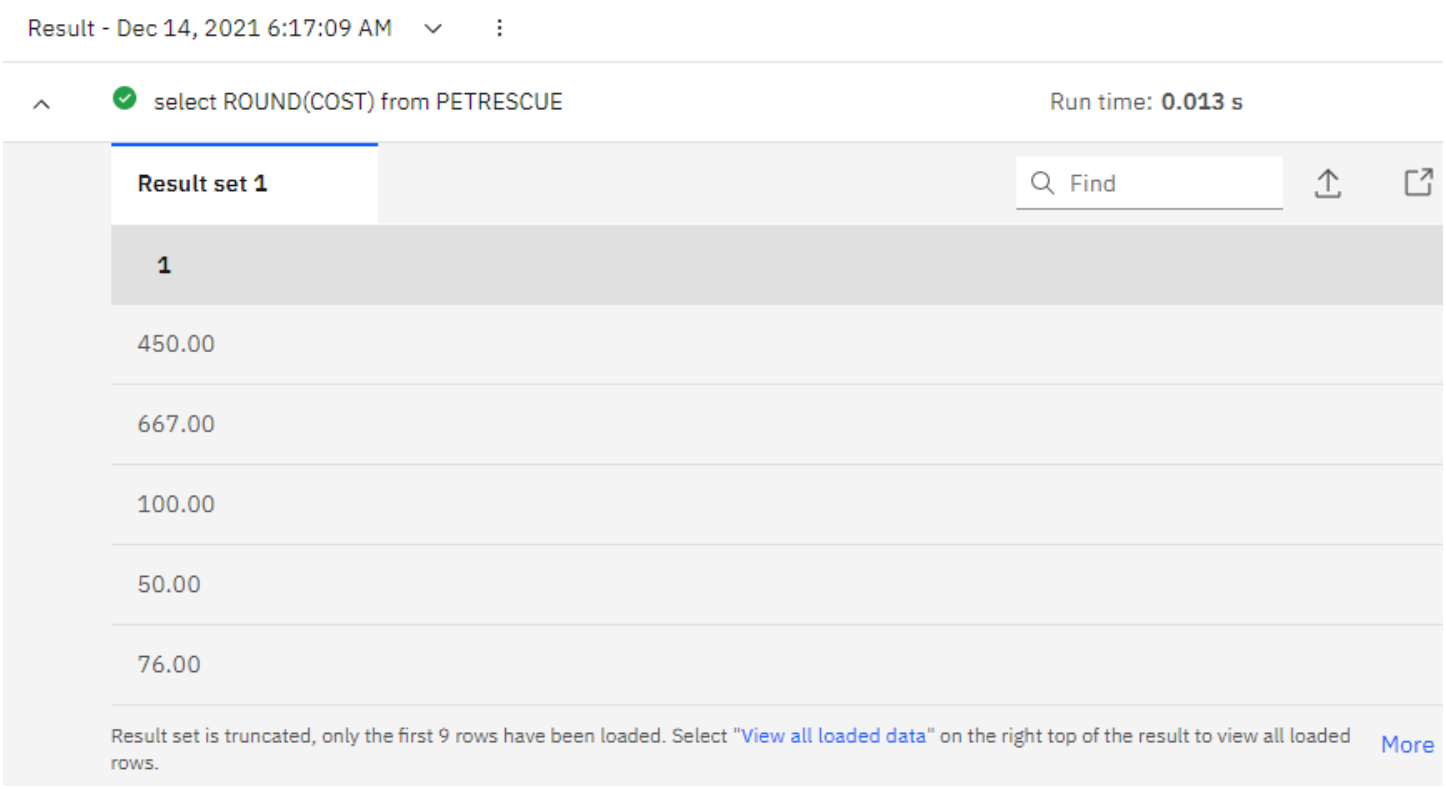
```
select AVG(COST/QUANTITY) from PETRESCUE where ANIMAL = 'Dog';
```



Exercise 3 Solutions: Scalar and String Functions

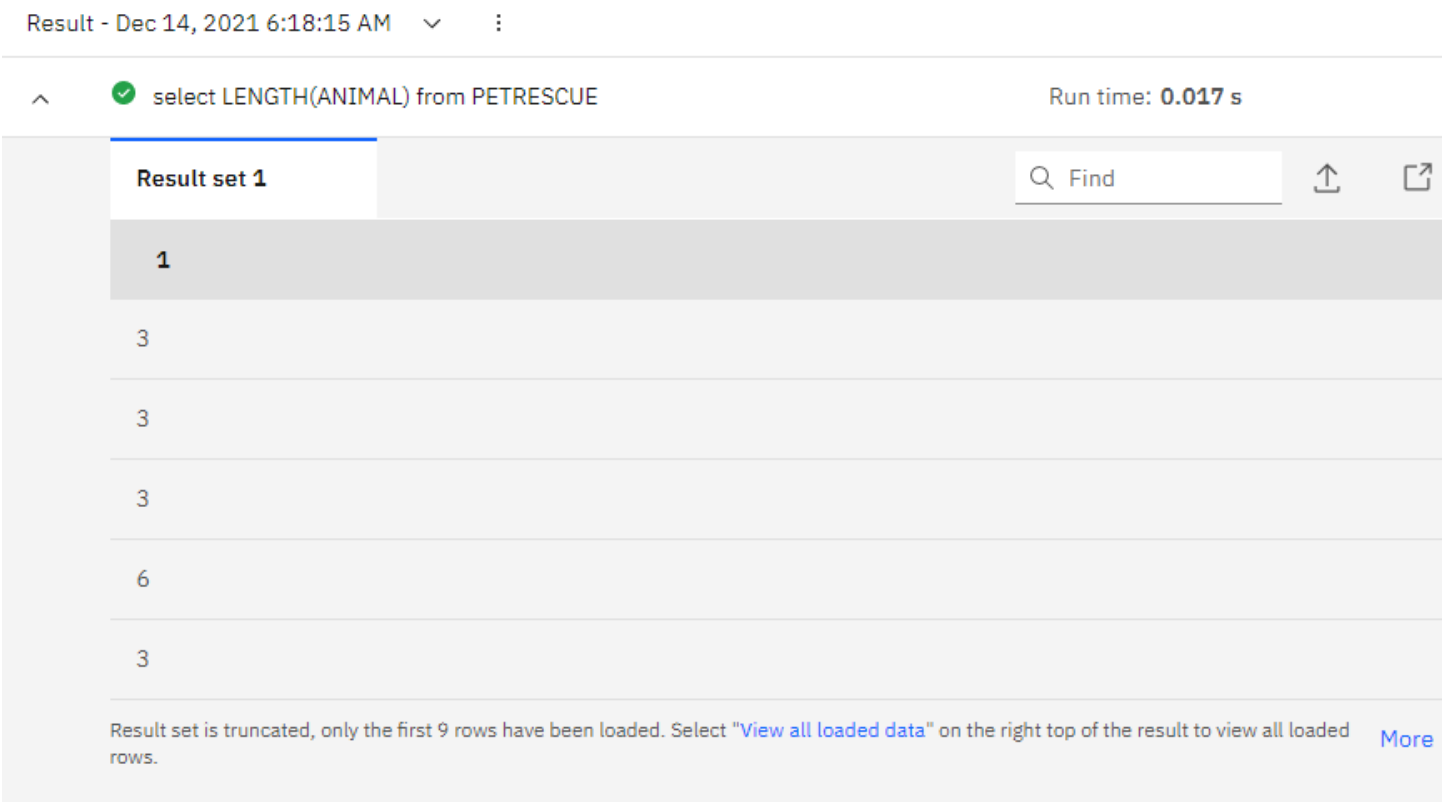
Query B1: Enter a function that displays the rounded cost of each rescue.

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



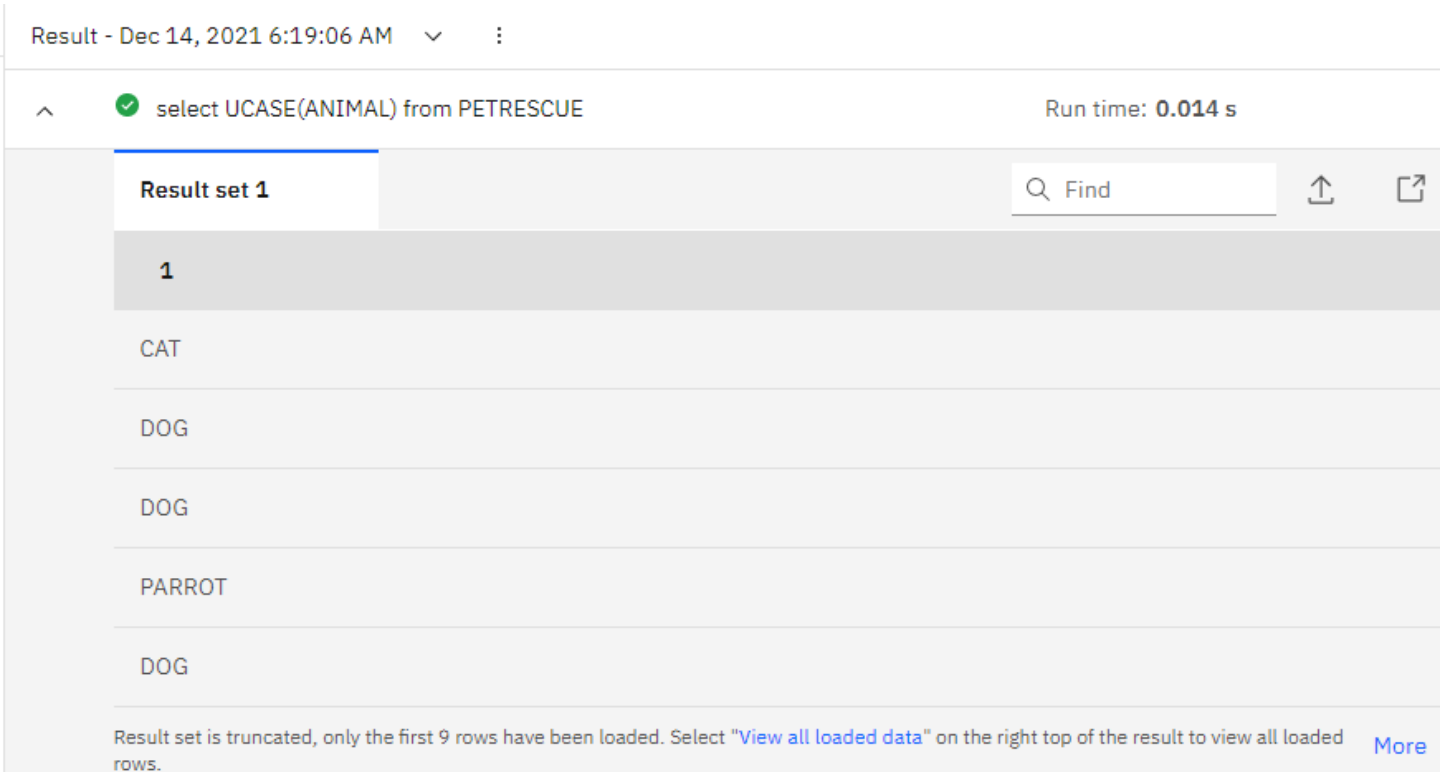
Query B2: Enter a function that displays the length of each animal name.

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



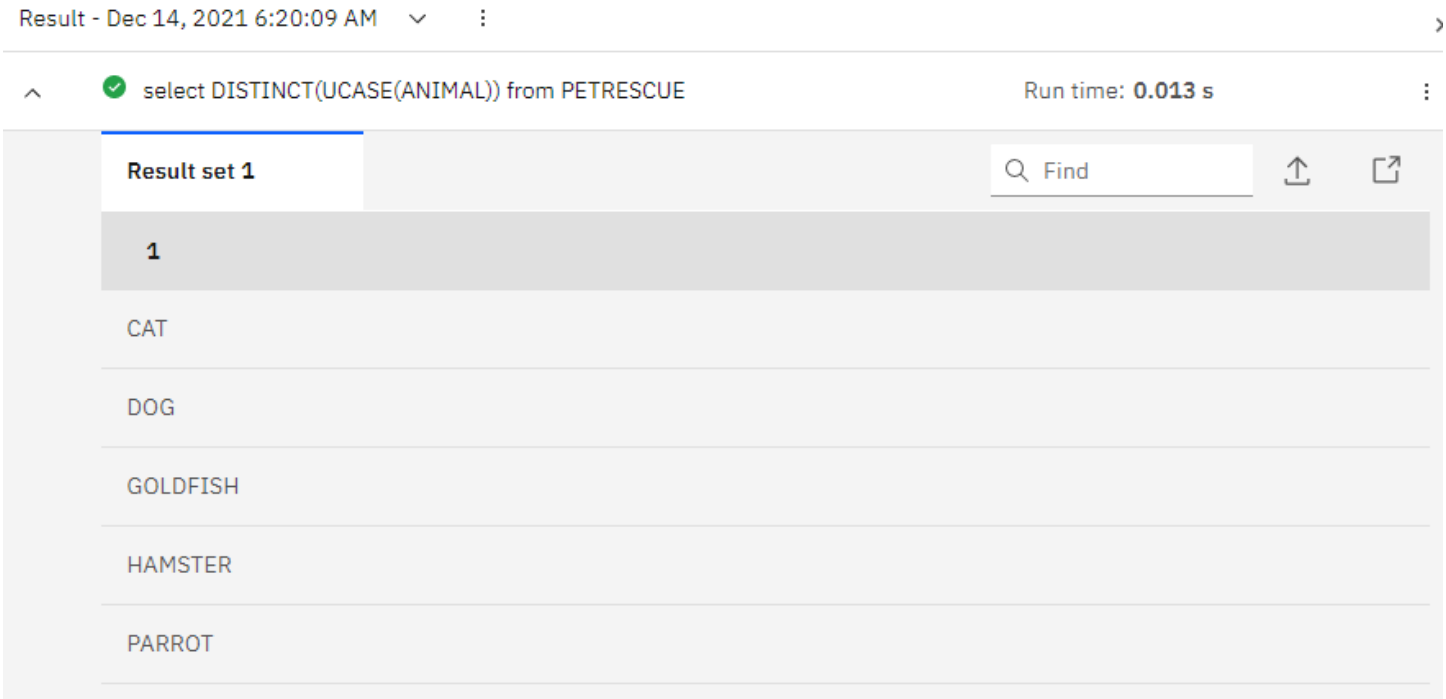
Query B3: Enter a function that displays the animal name in each rescue in uppercase.

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



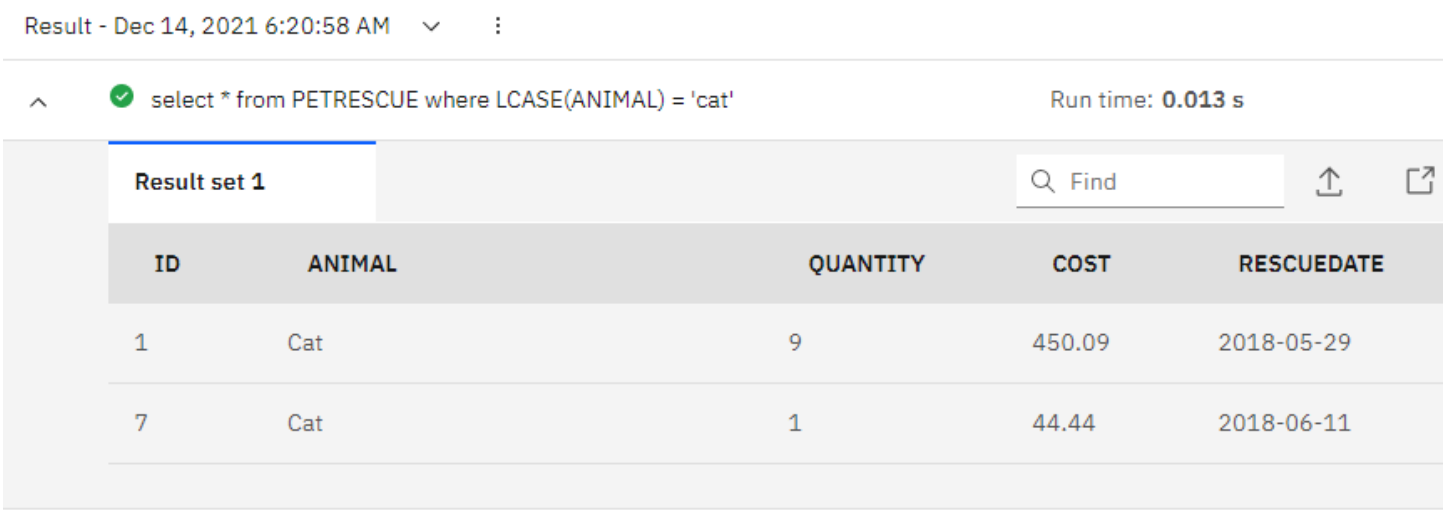
Query B4: Enter a function that displays the animal name in each rescue in uppercase without duplications.

```
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';
```

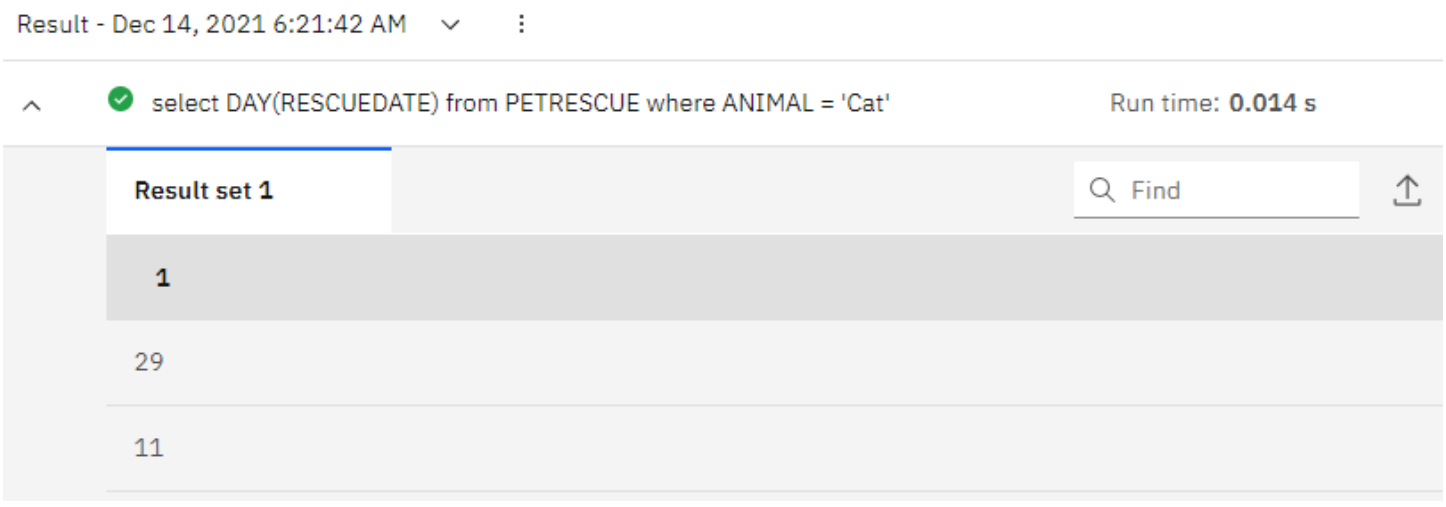


ID	ANIMAL	QUANTITY	COST	RESCUEDATE
1	Cat	9	450.09	2018-05-29
7	Cat	1	44.44	2018-06-11

Exercise 4 Solutions: Date and Time Functions

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

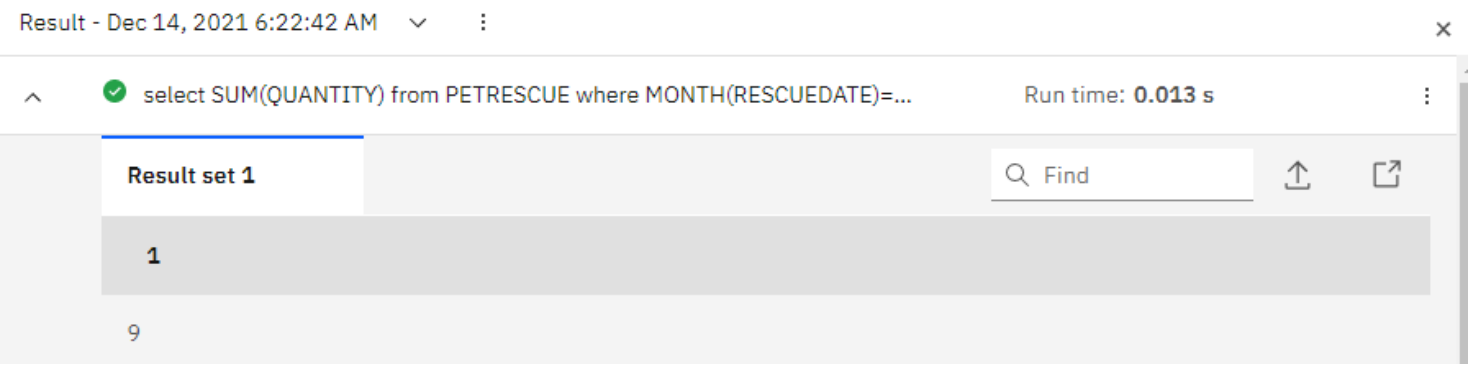
```
select DAY(RESCUEDATE) from PETRESCUE where ANIMAL = 'Cat';
```



1
29
11

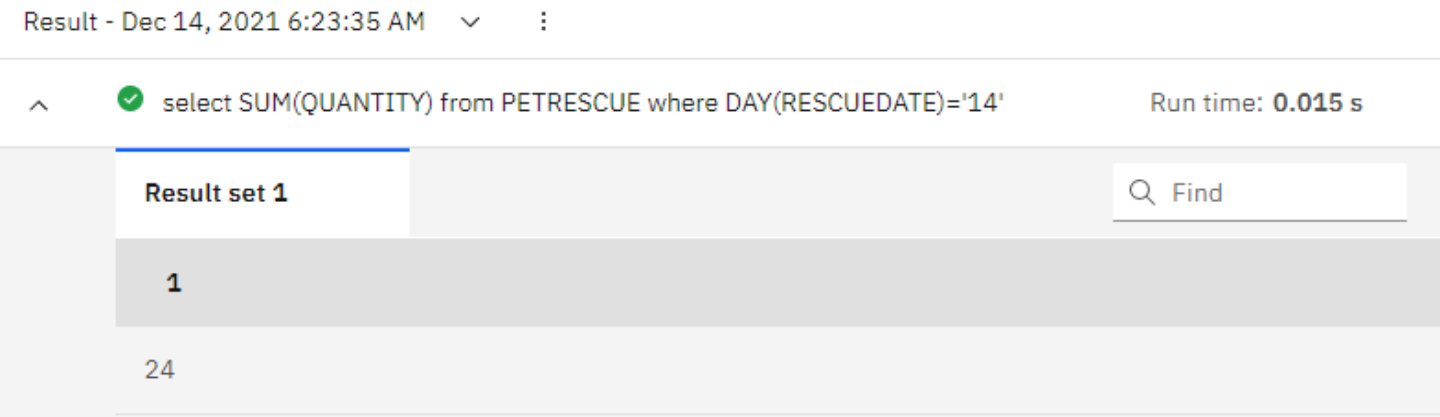
Query C2: Enter a function that displays the number of rescues on the 5th month.

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



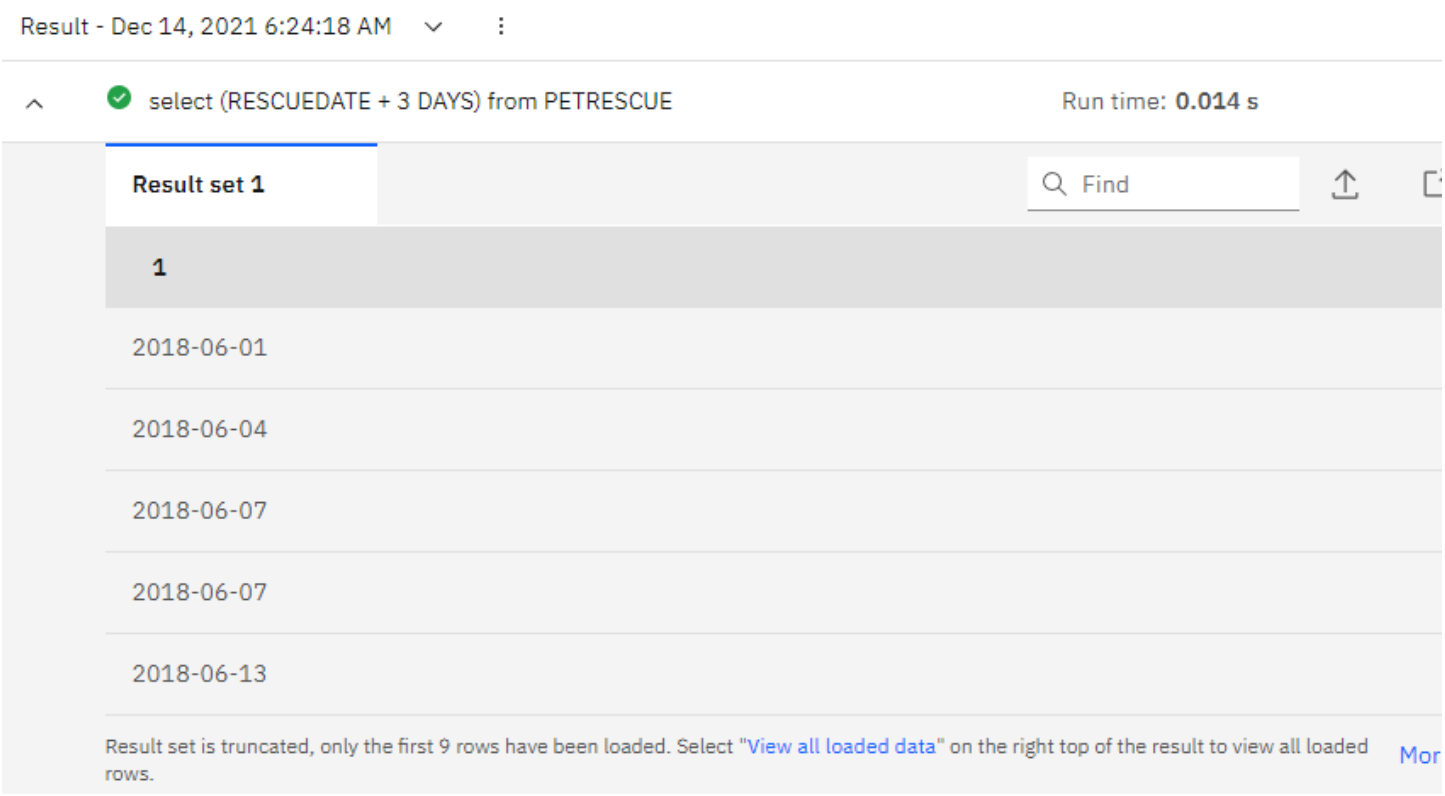
Query C3: Enter a function that displays the number of rescues on the 14th day of the month.

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



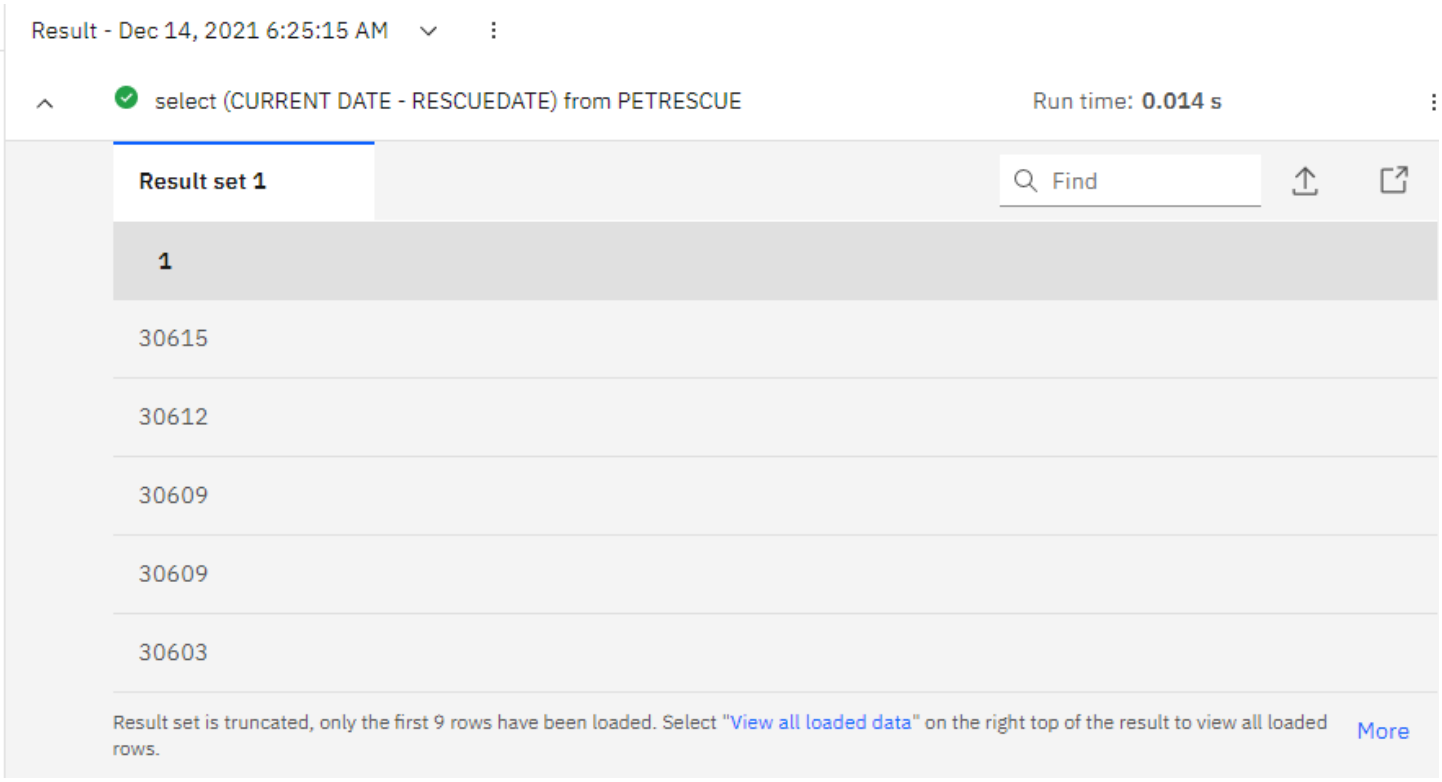
Query C4: Animals rescued should see the vet within three days of arrivals. Enter a function that displays the third day from each rescue.

```
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

In this video you'll learn how to write sub-queries or nested selects statements.

Sub-queries or sub selects are like regular queries but placed within parentheses

and nested inside another query.

This allows you to form more powerful queries than would have been otherwise possible.

An example of a nested query is shown. In this example, the sub-query is inside the where clause of another query.

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. Let's consider a scenario which may

necessitate the use of sub-queries. Let's say, we want to retrieve the list of employees

who earn more than the average salary. To do so, you could try this code.

Select * from employees, where salary > AVG(salary).

However, 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.

Select EMP_ID, F_NAME, L_NAME, SALARY from employees where SALARY < (select AVG (SALARY) from employees).

Notice that the average function is evaluated in the first part of the sub-query.

Allowing us to circumvent the limitation of evaluating it directly in the WHERE clause.

The sub-select doesn't just have to go in the WHERE clause. It can also go in other parts of

the query, such as in the list of columns to be selected. Such sub-queries are called column expressions.

Now, let's look at a scenario where we might want to use a column expression. Say we wanted to compare the salary of each employee with the average salary.

We could try a query like 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.

We can circumvent this error by using the average function in a sub-query placed in the list of the columns.

For example, 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. Sub-queries like these are sometimes called derived tables or table expressions.

Because the outer query uses the results of the sub-query as a data source.

Let's look at an example to create a table expression that contains nonsensitive employee information.

Select * from (select EMP_ID, F_NAME, L_NAME, DEP_ID from employees) AS EMP4ALL.

The derived table in a sub-query does not include sensitive fields like date of birth or salary.

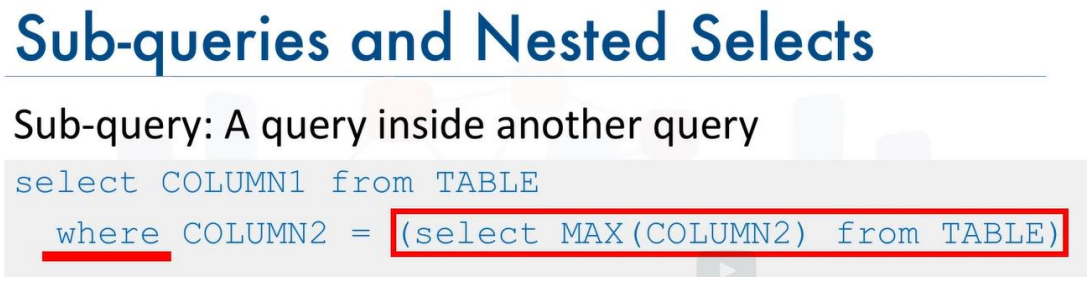
This example is a trivial one, and we could just as easily have included the columns in the outer query.

However, such derived tables can prove to be powerful in more complex situations such as when

working with multiple tables and doing joins. In this video, you have seen how sub-queries and nested queries can be used to form richer queries and how they can

overcome some of the limitations of aggregate functions.

You also learned to use sub-queries in the WHERE clause, in the list of columns and in the FROM clause.



Sub-queries and Nested Selects

Sub-query: A query inside another query

```
select COLUMN1 from TABLE
  where COLUMN2 = (select MAX(COLUMN2) from 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

Why use sub-queries?

To retrieve the list of employees who earn more than the average salary:

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

This query will result in error:

```
SQL0120N Invalid use of an aggregate function or
OLAP function.SQLCODE=-120, SQLSTATE=42903
```

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);
```

Sub-queries to evaluate Aggregate functions

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 ;
```

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 ;
```

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

Sub-queries in list of columns

Result:

EMP_ID	SALARY	AVG_SALARY
E1002	80000.00	68888.888888888888888888888
E1003	50000.00	68888.888888888888888888888
E1004	60000.00	68888.888888888888888888888
E1005	70000.00	68888.888888888888888888888
E1006	90000.00	68888.888888888888888888888
E1007	65000.00	68888.888888888888888888888
E1008	65000.00	68888.888888888888888888888
E1009	70000.00	68888.888888888888888888888
E1010	70000.00	68888.888888888888888888888

Sub-queries in FROM clause

- Substitute the TABLE name with a sub-query
- Called Derived Tables or Table Expressions

Sub-queries in FROM clause

- Substitute the TABLE name with a sub-query
- Called Derived Tables or Table Expressions

- Example:

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

Sub-queries in FROM clause

Result:

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

Summary

In this video you have learned:

- How sub-queries and nested queries form richer queries
- How they overcome limitations of aggregate functions
- How to use sub-queries in the:
  - WHERE clause
  - list of columns
  - FROM clause

Hands-on Lab : Sub-queries and Nested SELECTs

In this lab, you will run through some SQL practice problems that will provide hands-on experience with nested SQL SELECT statements (also known as Sub-queries).

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);
```

Software Used in this Lab

In this lab, you will use an IBM Db2 Database. Db2 is a Relational Database Management System (RDBMS) from IBM, designed to store, analyze and retrieve data efficiently.

To complete this lab you will utilize a Db2 database service on IBM Cloud. If you did not already complete this lab task earlier in this module, you will not yet have access to Db2 on IBM Cloud, and you will need to follow the lab below first:

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

Database Used in this Lab

The database used in this lab is an internal database. You will be working on a sample HR database. This HR database schema consists of 5 tables
called **EMPLOYEES**, **JOB_HISTORY**, **JOBS**, **DEPARTMENTS** and **LOCATIONS**. Each table has a few rows of sample data. The following diagram shows the tables for the HR database:

SAMPLE HR DATABASE TABLES

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

JOB_HISTORY

EMPL_ID	START_DATE	JOBS_ID	DEPT_ID
E1001	2000-01-30	100	2
E1002	2010-08-16	200	5
E1003	2016-08-10	300	5

JOBS

JOB_IDENT	JOB_TITLE	MIN_SALARY	MAX_SALARY
100	Sr. Architect	60000	100000
200	Sr.SoftwareDeveloper	60000	80000
300	Jr.SoftwareDeveloper	40000	60000

DEPARTMENTS

DEPT_ID_DEP	DEP_NAME	MANAGER_ID	LOC_ID
2	Architect Group	30001	L0001
5	Software Development	30002	L0002
7	Design Team	30003	L0003
5	Software	30004	L0004

LOCATIONS

LOCT_ID	DEP_ID_LOC
L0001	2
L0002	5
L0003	7

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

Objectives

After completing this lab you will be able to:

- Write SQL queries that demonstrate the necessity of using sub-queries
- Compose sub-queries in the where clause
- Build Column Expressions (i.e. sub-query in place of a column)
- Write Table Expressions (i.e. sub-query in place of a table)

Instructions

When you approach the exercises in this lab, follow the instructions to run the queries on Db2:

- Go to the Resource List of IBM Cloud by logging in 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, open the Db2 Console by clicking on **Open Console** button. Click on the 3-bar menu icon in the top left corner and go to the **Run SQL** page. The Run SQL tool enables you to run SQL statements.
    - If needed, follow Hands-on Lab : Sign up for IBM Cloud, Create Db2 service instance and Get started with the Db2 console

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

▼ Solution

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

▼ Output

--- Query 1 --- select * from employees where salary… Run time: **0.011 s**

Status: **Failed**

Error message

Invalid use of an aggregate function or OLAP function.. SQLCODE=-120, SQLSTATE=42903, DRIVER=4.26.14

Learn more about this error

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);
```

--- Query 2--- select EMP_ID, F_NAME, L_NAME… Run time: **0.001 s**

Result set 1 *Search*

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

Show Less

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.

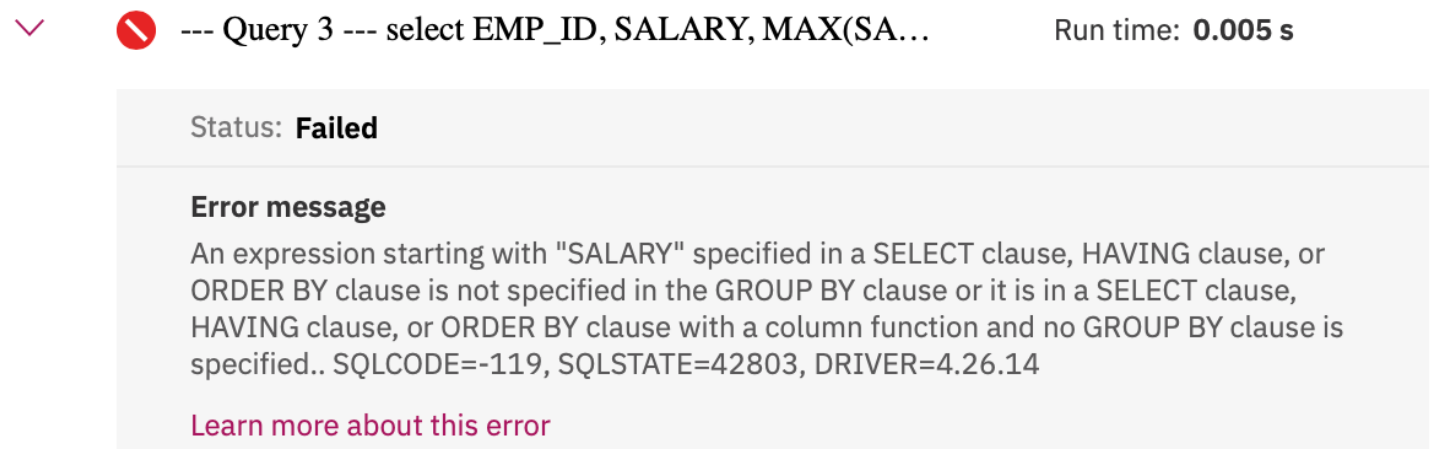
Hint

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

--- Query 4 --- select EMP_ID, SALARY, (select M… Run time: **0.001 s**

Result set 1 Search

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

5.Problem:

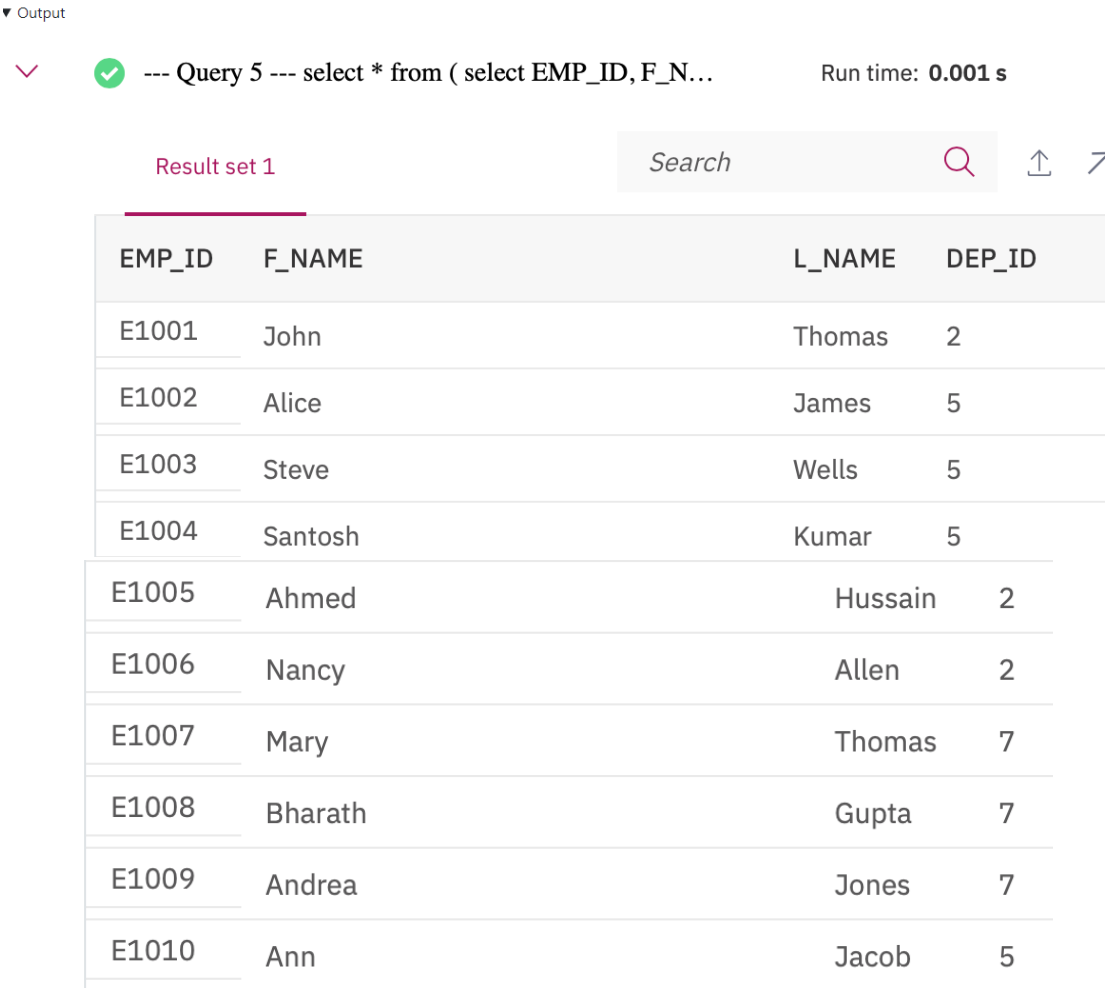
Execute a Table Expression for the EMPLOYEES table that excludes columns with sensitive employee data (i.e. does not include columns: SSN, B_DATE, SEX, ADDRESS, SALARY).

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;
```



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

Run SQL

* SubQueries... ×

```sql

--- Query 1 ---
select *
from employees
where salary < AVG(salary);


--- Query 2---
select EMP_ID, F_NAME, L_NAME, SALARY
from employees
where SALARY <
    (select AVG(SALARY)
    from employees)
;

--- Query 3 ---
select EMP_ID, SALARY, MAX(SALARY) AS MAX_SALARY
from employees ;


--- Query 4 ---
select EMP_ID, SALARY, ( select MAX(SALARY) from employees ) AS MAX_SALARY
from employees
;

--- Query 5 ---
select *
from ( select EMP_ID, F_NAME, L_NAME, DEP_ID from employees) AS EMP4ALL
;

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

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

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

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

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

Working with Multiple Tables

In this video, you will learn how to write queries that access more than one table.

There are several ways to access multiple tables in the same query.

Namely, using Sub-queries, Implicit JOIN,

and JOIN operators, such as INNER JOIN and OUTER JOIN.

In this video, we'll examine the first two options.

The third option is covered in more detail in other videos. Let's consider the employees and departments tables from a previous video.

The employees table contains several columns for categories, such as employee ID,

first name, last name, and salary to name a few. The Departments table contains a department ID, department name, Manager ID, and location ID.

Some sample data from these tables is shown here.

We will utilize these tables for the examples in this video.

In a previous video, we learned how to use sub-queries.

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.

Let's say we want to retrieve only the list of employees from a specific location.

We do not have any location information in the employees table, but the departments table has a column called location ID. Therefore, we can use a sub-query from the Departments table as input to the employee table query as follows.

Select star from employees, where department_ID IN, select department_ID_department from departments, where location ID equals L0002.

Now, let's retrieve the department ID and department name for employees who earn more than $70,000. To do so, we will need a sub-query on the employees table to satisfy the salary criteria, and then feed it as input to an outer query on the departments table in order to get the matching department info.

Select department_ID_department department_name from departments, where department_ID_department IN, select department_ID from employees where salary is greater than 70,000.

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

Consider the example, select star from employees, departments.

Here we specify two tables in the FROM clause.

This results in a table join, but note we are not explicitly using the join operator.

The resulting join in this example is called a full join or Cartesian join,

because every row in the first table is joined with every row in the second table.

If you examine the results set, you will see more rows than in both tables individually.

We can use additional operands to limit the result set. Let's look at an example where we limit the result set to only rows with matching department IDs.

Select star from employees, departments, where employees department_ID equals departments, department_ID_department.

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.

Select star from employees E, departments D, where E department_ID equals D department_ID_department.

Here, we define the alias E for employees table and D for departments table,

and then use these aliases in the WHERE clause.

If we wanted to see the department name for each employee, we would enter the code as follows:

Select Employee_ID, Department_Name

from employees E, departments D,

where E department_ID equals D

department_ID_department.

Similar to before, the column names in the select clause can also be prefixed by aliases as shown in the query.

Select E. Employee_ID, D. Department_ID_department,

from employees E. Departments D where

E.Department_ID equals D. Department_ID_department.

QUERYING 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 );
```

Accessing Multiple Tables with Sub-queries

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' );
```

Multiple Tables with Sub-queries

Result:

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

Multiple Tables with Sub-queries

Result:

DEPT_ID_DEP	DEP_NAME
5	Software Group

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

Accessing multiple tables with Implicit Join

Use additional operands to limit the result set:

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

Use shorter aliases for table names:

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

Accessing multiple tables with Implicit Join

Query:

```
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

Accessing multiple tables with Implicit Join

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;
```

Accessing multiple tables with Implicit Join

Query:

```
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

Accessing multiple tables with Implicit Join

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
```

Hands-on Lab: Working with Multiple Tables

Estimated time needed: 30 minutes

In this lab, you will through some SQL practice problems that will provide hands-on experience with SQL queries that access multiple tables. You will be:

- Accessing Multiple Tables with Sub-Queries
- Accessing Multiple Tables with Implicit Joins

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;
```

Software Used in this Lab

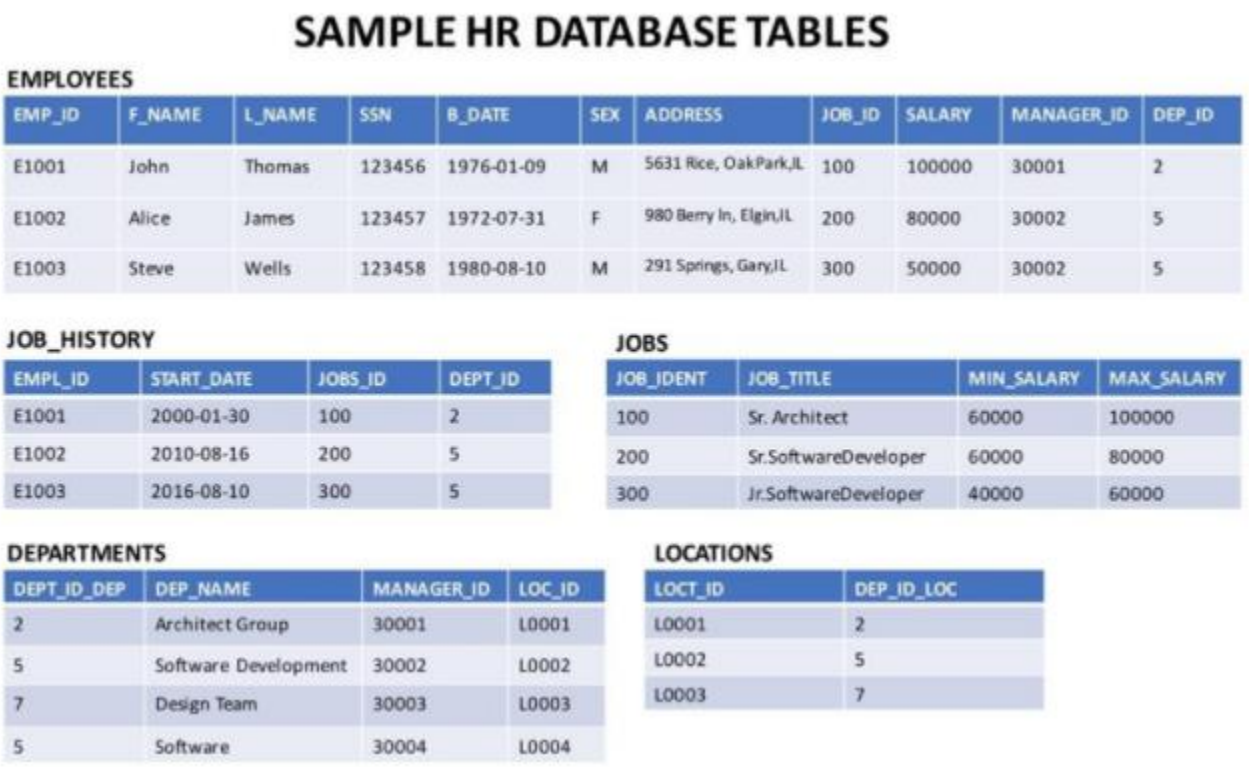
In this lab, you will use IBM Db2 Database. Db2 is a Relational Database Management System (RDBMS) from IBM, designed to store, analyze and retrieve the data efficiently.

To complete this lab you will utilize a Db2 database service on IBM Cloud. If you did not already complete this lab task earlier in this module, you will not yet have access to Db2 on IBM Cloud, and you will need to follow the lab below first:

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

Database Used in this Lab

The database used in this lab is an internal database. You will be working on a sample HR database. This HR database schema consists of 5 tables
called **EMPLOYEES**, **JOB_HISTORY**, **JOBS**, **DEPARTMENTS** and **LOCATIONS**. Each table has a few rows of sample data. The following diagram shows the tables for the HR database:



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

Objectives

After completing this lab you will be able to:

- Write SQL queries that access more than one table
- Compose queries that access multiple tables using a nested statement in the WHERE clause
- Build queries with multiple tables in the FROM clause
- Write Implicit Join queries with join criteria specified in the WHERE clause
- Specify aliases for table names and qualify column names with table aliases

Instructions

When you approach the exercises in this lab, follow the instructions to run the queries on Db2:

- Go to the Resource List of IBM Cloud by logging in 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, open the Db2 Console by clicking on **Open Console** button. Click on the 3-bar menu icon in the top left corner and go to the **Run SQL** page. The Run SQL tool enables you to run SQL statements.
    - If needed, follow Hands-on Lab : Sign up for IBM Cloud, Create Db2 service instance and Get started with the Db2 console

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

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

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

Result set 1

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

Result set 1

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

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

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;
```

Result set 1

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

Hands-on Lab: Working with Multiple Tables

Query 1A ---

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

;

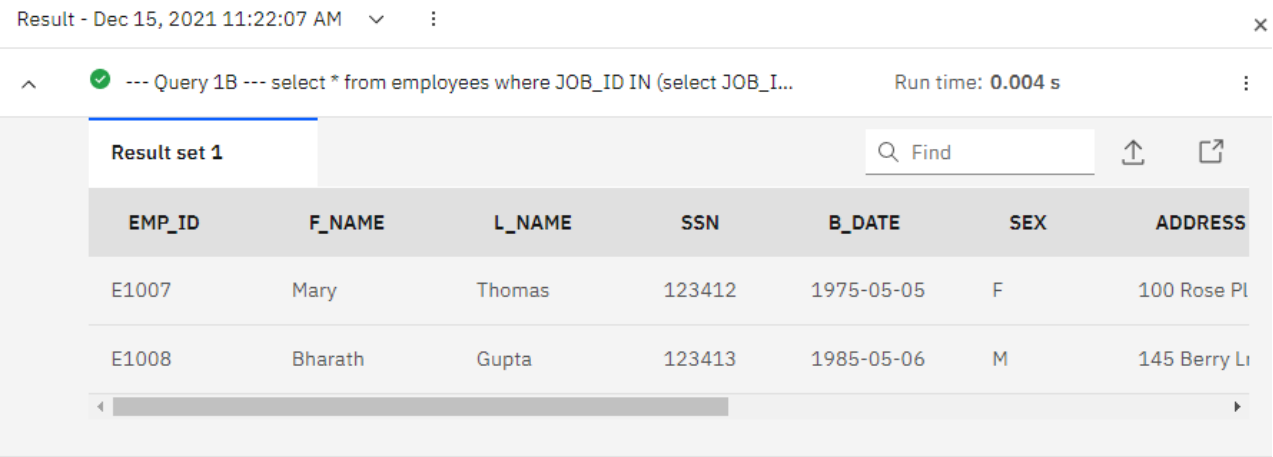
EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS
E1002	Alice	James	123457	1972-07-31	F	980 Berry ln
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs
E1004	Santosh	Kumar	123459	1985-07-20	M	511 Aurora ,
E1005	Ahmed	Hussain	123410	1981-01-04	M	216 Oak Tre
E1006	Nancy	Allen	123411	1978-02-06	F	111 Green F

Result set is truncated, only the first 9 rows have been loaded. Select "View all loaded data" on the right top of the result to view all loaded rows. More

--- Query 1B ---

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

;



EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS
E1007	Mary	Thomas	123412	1975-05-05	F	100 Rose Pl
E1008	Bharath	Gupta	123413	1985-05-06	M	145 Berry Li

--- Query 1C ---

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

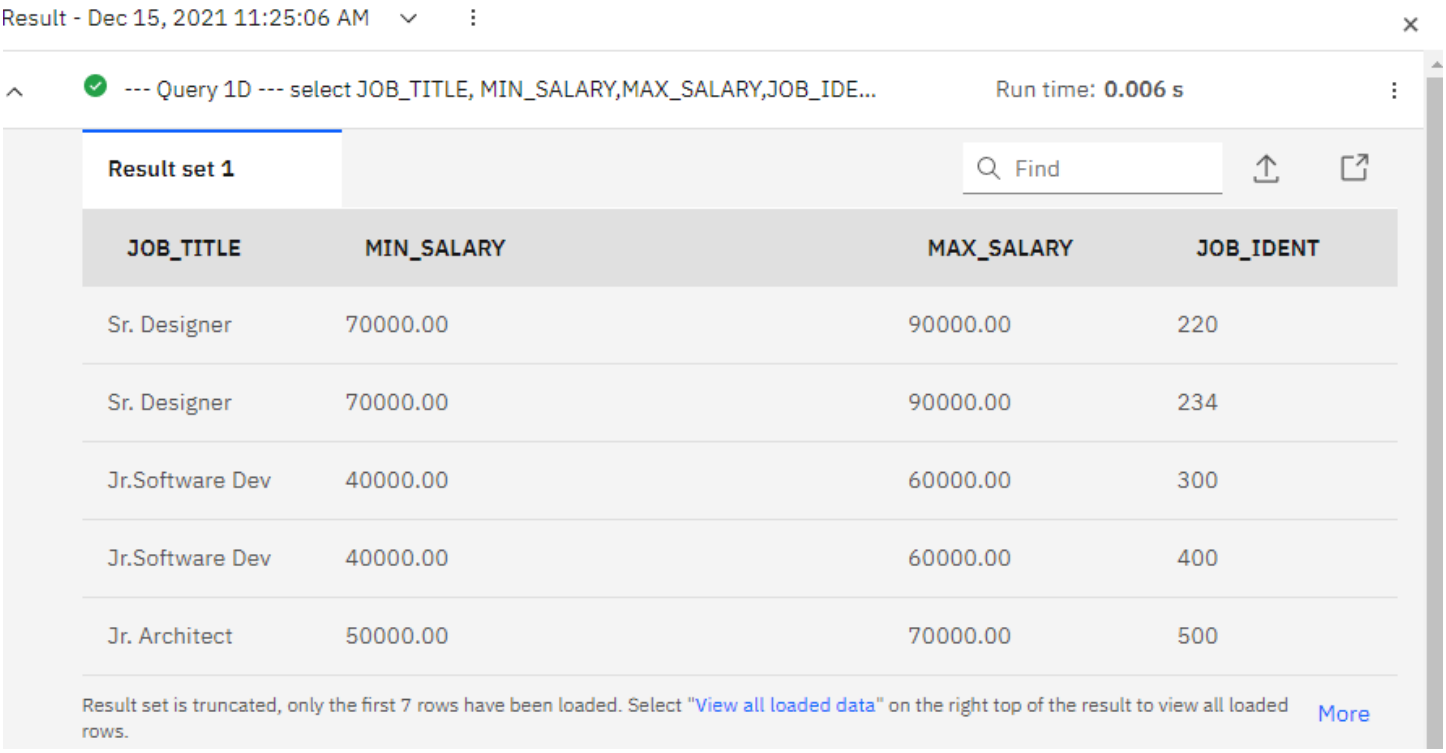
;

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

--- Query 1D ---

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)

;



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

Result set is truncated, only the first 7 rows have been loaded. Select "View all loaded data" on the right top of the result to view all loaded rows. More

--- Query 1E ---

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')

;

Result - Dec 15, 2021 11:26:33 AM

--- Query 1E --- select JOB_TITLE, MIN_SALARY,MAX_SALARY,JOB_IDE... Run time: **0.040 s**

Result set 1

Find

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

--- Query 2A ---

select * from employees, jobs

;

Result - Dec 15, 2021 11:27:21 AM

--- Query 2A --- select * from employees, jobs Run time: **0.005 s**

Result set 1

Find

EMP_ID	F_NAME	L_NAME	SSN	B_DATE	SEX	ADDRESS
E1002	Alice	James	123457	1972-07-31	F	980 Berry ln
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs
E1004	Santosh	Kumar	123459	1985-07-20	M	511 Aurora ,
E1005	Ahmed	Hussain	123410	1981-01-04	M	216 Oak Tre
E1006	Nancy	Allen	123411	1978-02-06	F	111 Green F

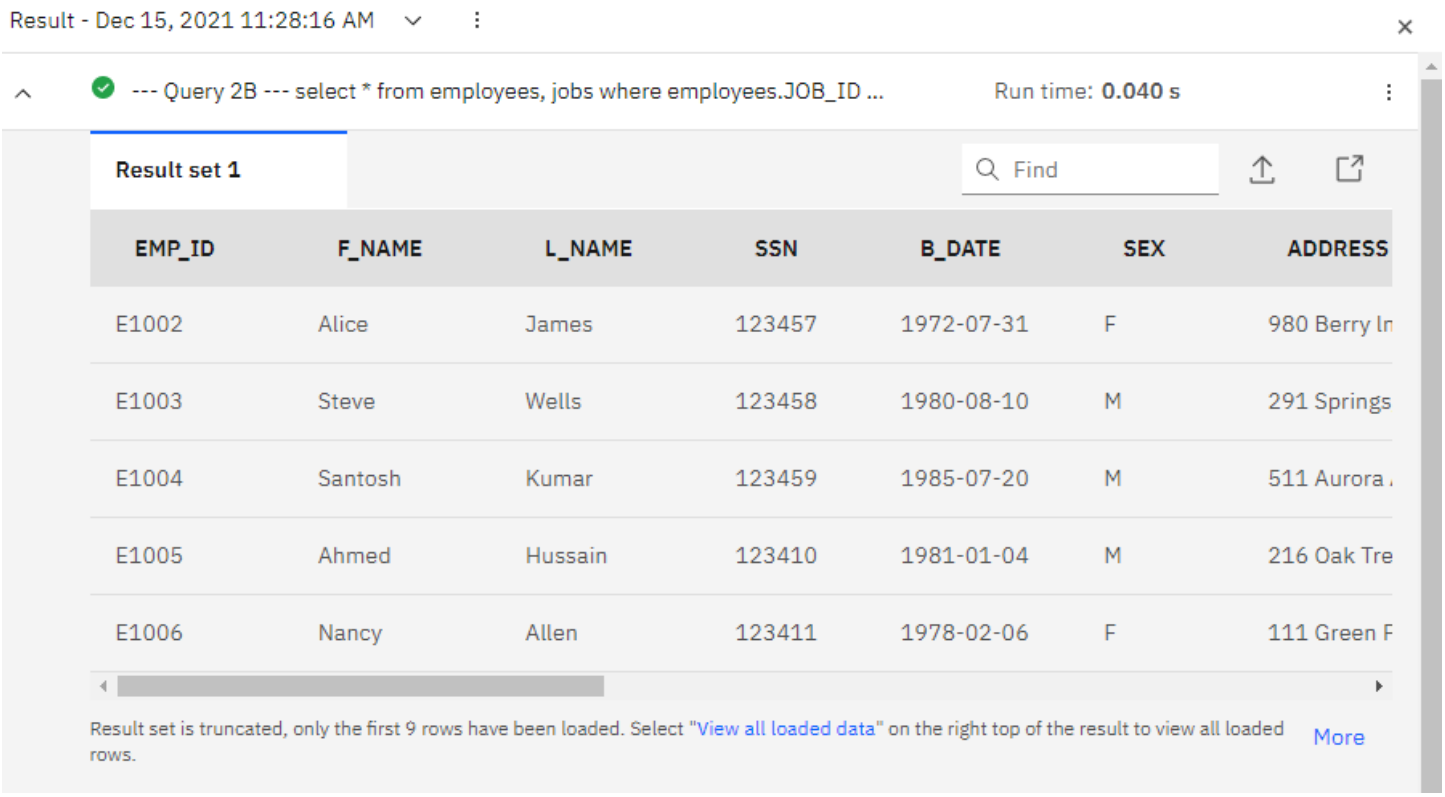
Result set is truncated, only the first 81 rows have been loaded. Select "View all loaded data" on the right top of the result to view all loaded rows.

More

--- Query 2B ---

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

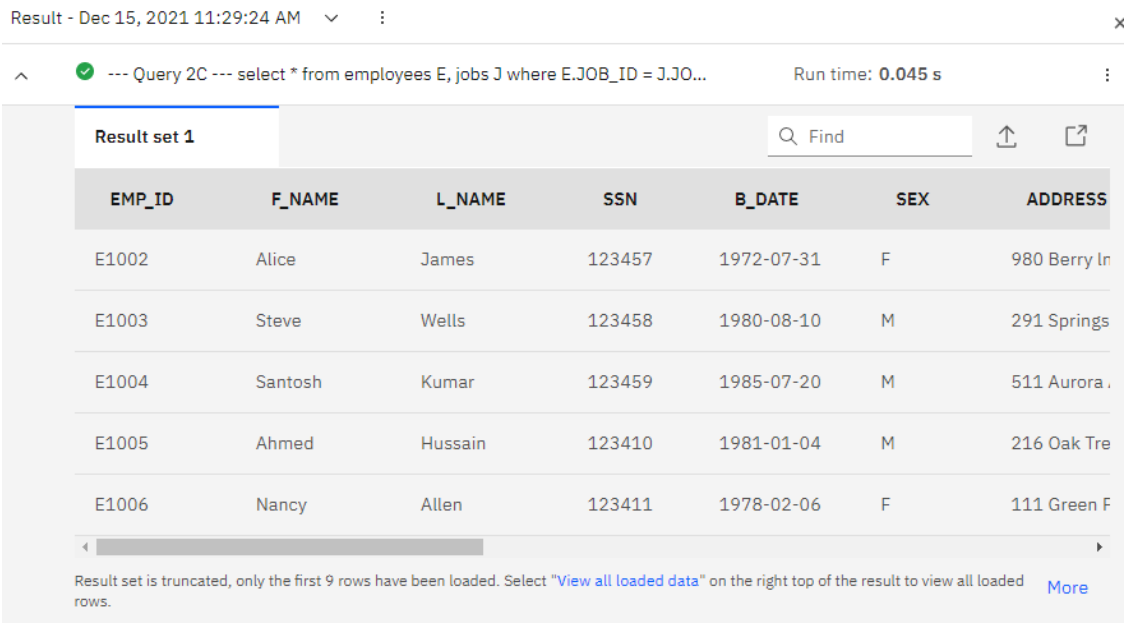
;



--- Query 2C ---

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

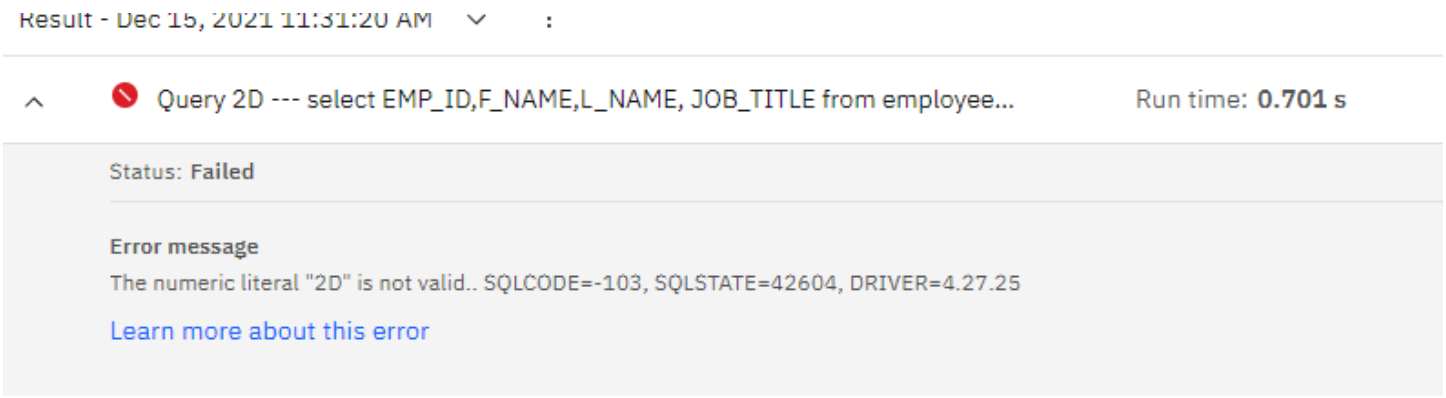
;



Query 2D ---

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

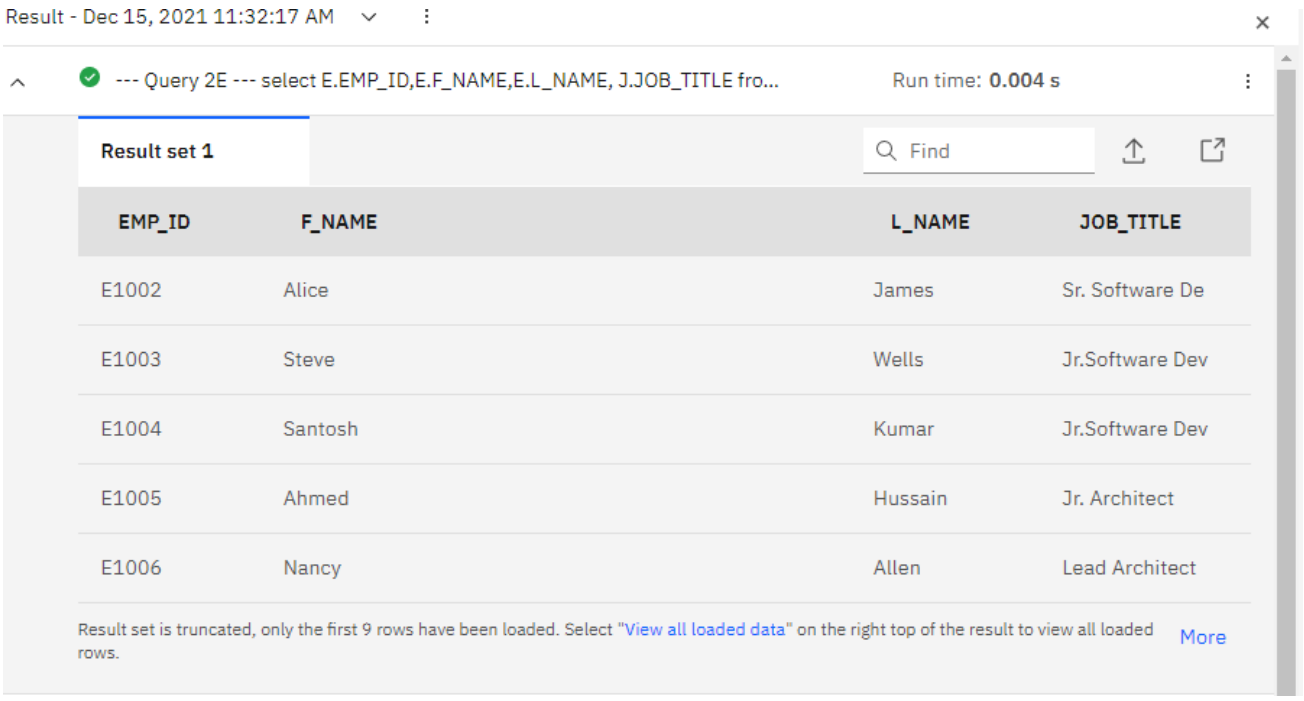
;



--- Query 2E ---

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

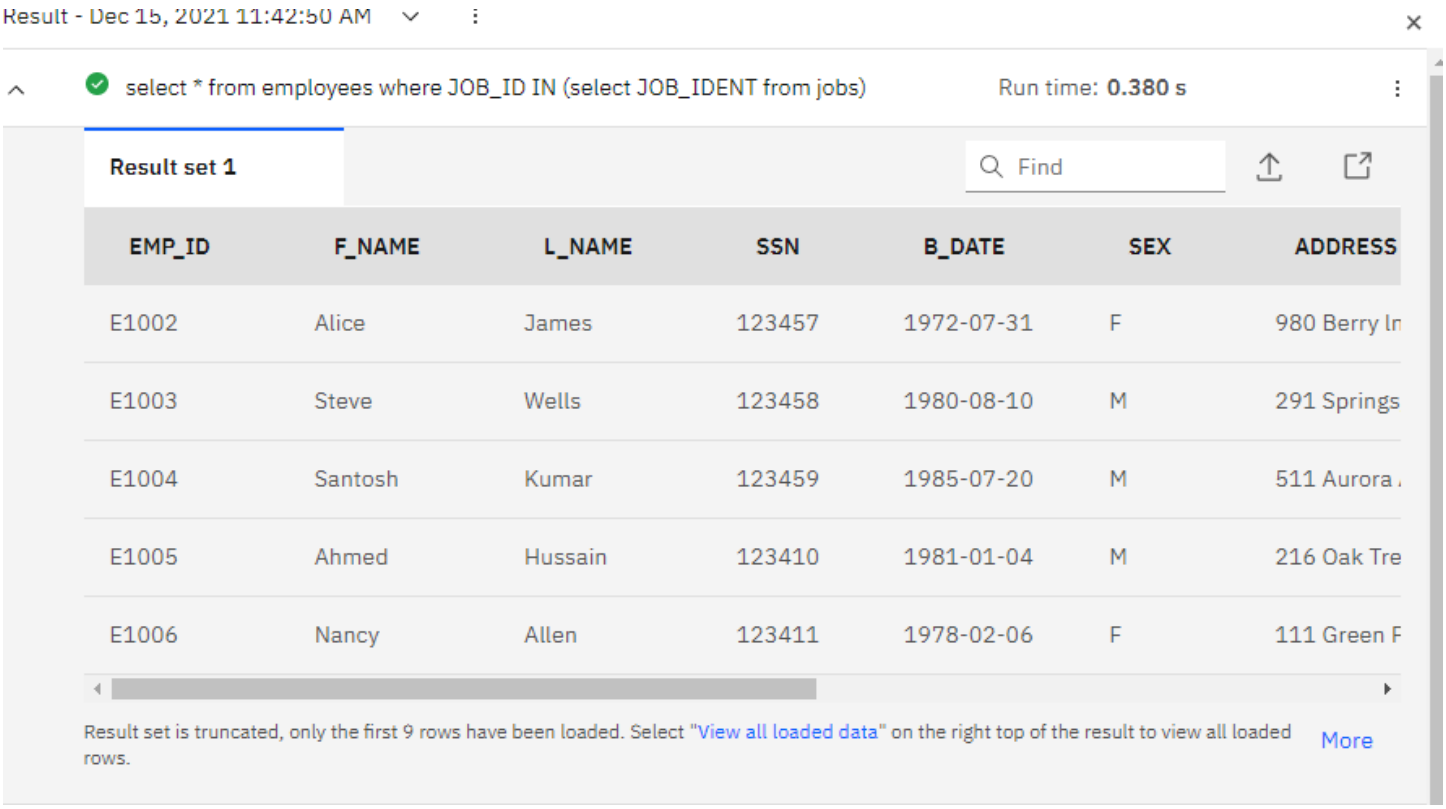
;



1.Problem:

Retrieve only the EMPLOYEES records that correspond to jobs in the JOBS tab

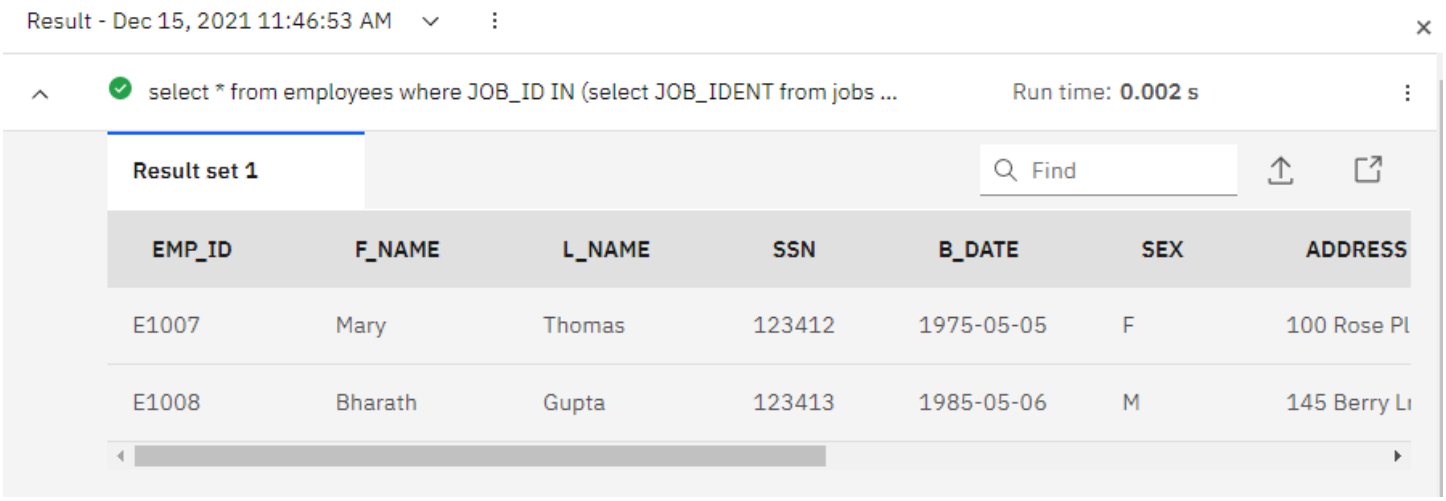
select * from employees where JOB_ID IN (select JOB_IDENT from jobs);



2. Problem

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

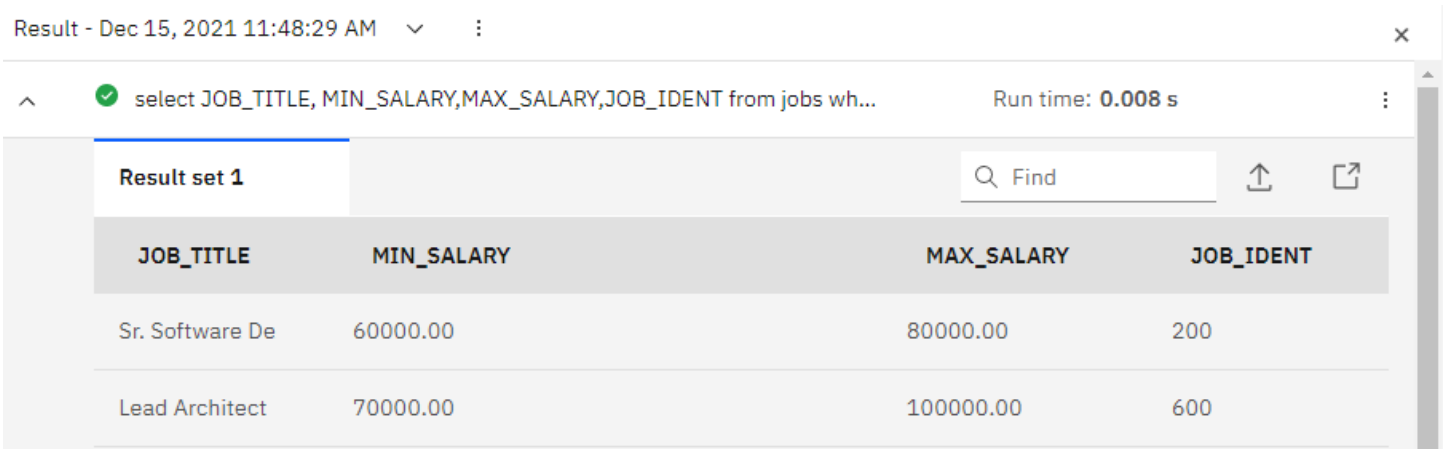
select * from employees where JOB_ID IN (select JOB_IDENT from jobs where JOB_TITLE= 'Jr. Designer');



3. Problem

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

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



4. Problem

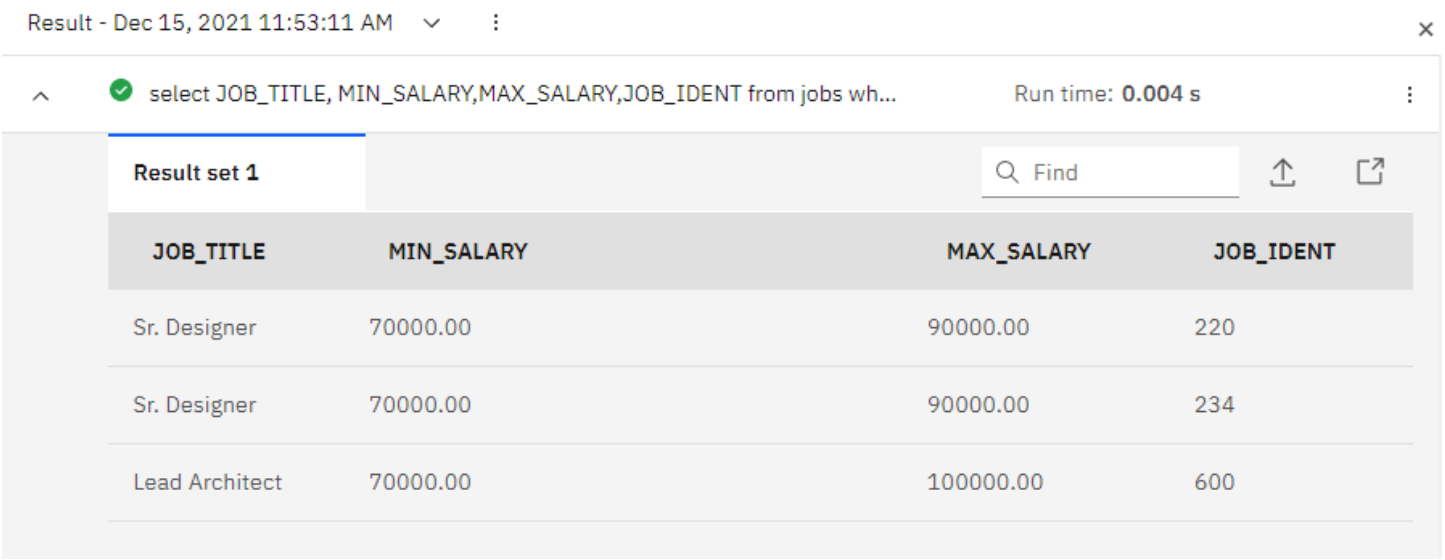
Retrieve JOB information and list of employees whose birth year is after 1976.

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);

5. Problem

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

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');

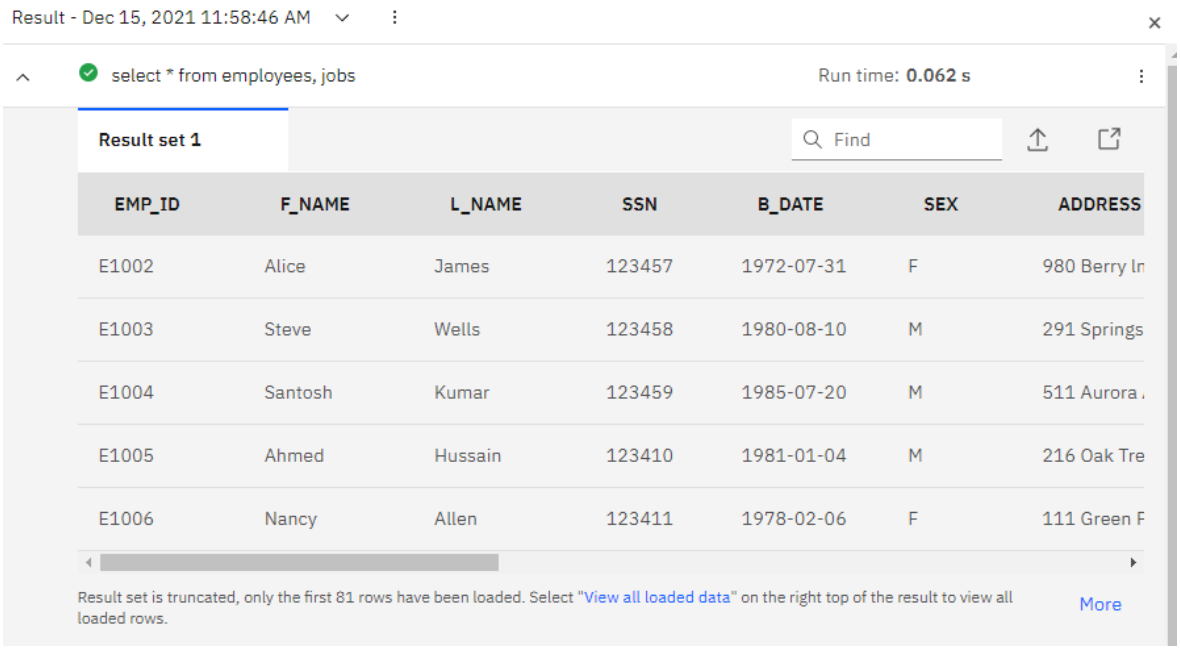


Exercise 2: Accessing Multiple Tables with Implicit Joins

1.Problem

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

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

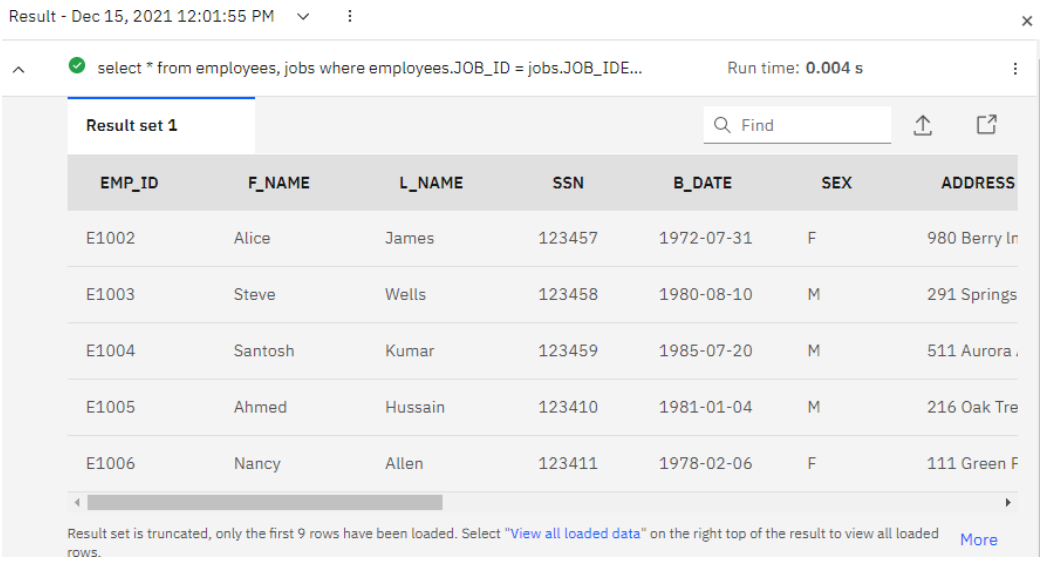


2.Problem

> *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;
```

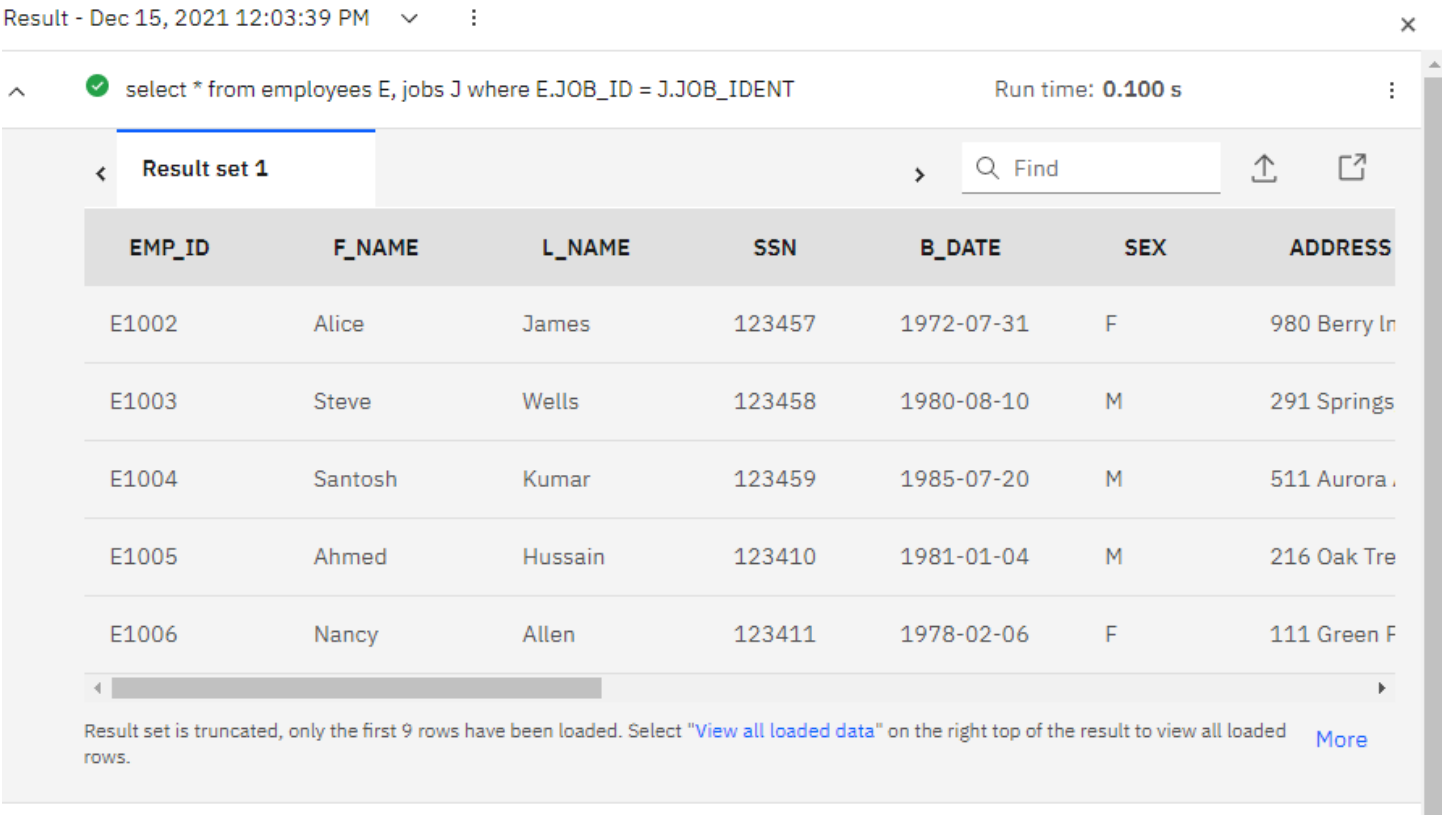


3. Problem:

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

▼ Solution

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

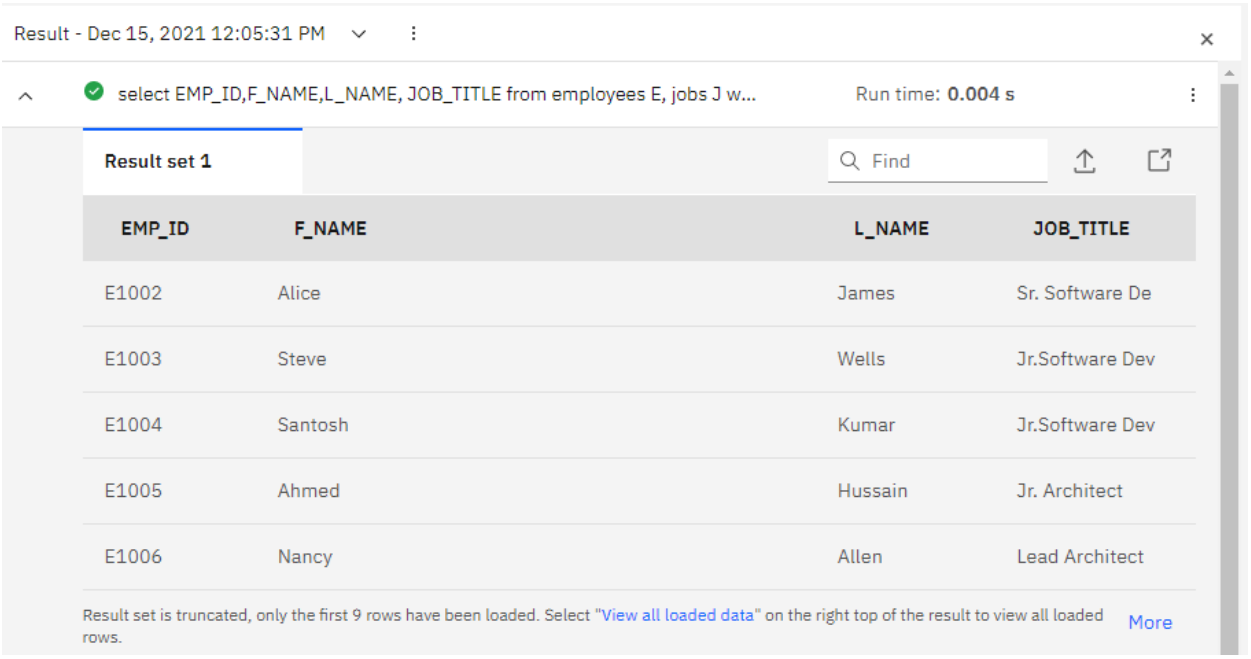


4. Problem:

> *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;
```



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;
```

Result - Dec 15, 2021 12:07:08 PM

select E.EMP_ID,E.F_NAME,E.L_NAME, J.JOB_TITLE from employees E, j... Run time: **0.006 s**

Result set 1

Find

EMP_ID	F_NAME	L_NAME	JOB_TITLE
E1002	Alice	James	Sr. Software De
E1003	Steve	Wells	Jr.Software Dev
E1004	Santosh	Kumar	Jr.Software Dev
E1005	Ahmed	Hussain	Jr. Architect
E1006	Nancy	Allen	Lead Architect

Result set is truncated, only the first 9 rows have been loaded. Select "View all loaded data" on the right top of the result to view all loaded rows. More

```
1  ;
2  select * from employees where JOB_ID IN (select JOB_IDENT from jobs);
3  select * from employees where JOB_ID IN (select JOB_IDENT from jobs where JOB_TITLE= 'Jr. Designe
4  select JOB_TITLE, MIN_SALARY,MAX_SALARY,JOB_IDENT from jobs where JOB_IDENT IN (select JOB_ID fro
5  select JOB_TITLE, MIN_SALARY,MAX_SALARY,JOB_IDENT from jobs where JOB_IDENT IN (select JOB_ID fro
6  select JOB_TITLE, MIN_SALARY,MAX_SALARY,JOB_IDENT from jobs where JOB_IDENT IN (select JOB_ID fro
7  select * from employees, jobs;
8  select * from employees, jobs where employees.JOB_ID = jobs.JOB_IDENT;
9  select * from employees E, jobs J where E.JOB_ID = J.JOB_IDENT;
0  select EMP_ID,F_NAME,L_NAME, JOB_TITLE from employees E, jobs J where E.JOB_ID = J.JOB_IDENT;
1  select E.EMP_ID,E.F_NAME,E.L_NAME, J.JOB_TITLE from employees E, jobs J where E.JOB_ID = J.JOB_ID
```

Summary & Highlights

Bookmarked

Congratulations! You have completed this module. At this point in the course, you know:

- Most databases come with built-in functions that you can use in SQL statements to perform operations on data within the database itself.
- When you work with large datasets, you may save time by using built-in functions rather than first retrieving the data into your application and then executing functions on the retrieved data.
- You can use sub-queries to form more powerful queries than otherwise.
- You can use a sub-select expression to evaluate some built-in aggregate functions like the average function.
- Derived tables or table expressions are sub-queries where the outer query uses the results of the sub-query as a data source.

Practice Quiz

Bookmark this page

Question 1

1 point possible (ungraded)

Which of the following statements about built-in database functions is correct?

- ◯ Built-in database functions may increase network bandwidth consumed.
- ◯ Built-in database functions reduce the amount of data that is retrieved.
- ◯ Built-in database functions may increase processing time.
- ◯ Built-in database functions must be called from a programming language like Python.

Submit

Question 2

1 point possible (ungraded)

Which of the following SQL queries would return the day of the week each dog was rescued?

- ◯ SELECT RescueDate From PetRescue WHERE Animal = 'Dog';
- ◯ SELECT DAYOFWEEK(RescueDate) From PetRescue;
- ◯ SELECT DAY(RescueDate) From PetRescue WHERE Animal = 'Dog';
- ◯ SELECT DAYOFWEEK(RescueDate) From PetRescue WHERE Animal = 'Dog';

Submit

Question 3

1 point possible (ungraded)

What is the result of the following query: SELECT (Current_Date – RescueDate) FROM PetRescue

- ◯ Returns today's date.
- ◯ Returns the current date and rescue date columns.
- ◯ Returns how long it has been since each rescue.
- ◯ Returns the rescue date for each rescue.

Submit

Question 4

1 point possible (ungraded)

Which of the following queries will return the employees who earn less than the average salary?

- SELECT AVG(Salary) FROM Employees WHERE Salary < AVG(Salary)
- SELECT * FROM Employees WHERE Salary < AVG(Salary)
- SELECT * FROM Employees WHERE Salary < (SELECT AVG(Salary))
- SELECT * FROM Employees WHERE Salary < (SELECT AVG(Salary) FROM Employees);

Submit

Question 5

1 point possible (ungraded)

What are the three ways to work with multiple tables in the same query?

- Sub-queries, APPEND, JOIN operators
- Sub-queries, Implicit joins, JOIN operators
- Built-in functions, implicit joins, JOIN operators
- Sub-queries, Implicit joins, normalization

Module 3 Graded Quiz 2

Bookmark this page

Graded Quiz due Dec 23, 2021 19:09 +08

Question 1

1 point possible (graded)

Which of the following queries will return the data for employees who belong to the department with the highest value of department ID.

- SELECT * FROM EMPLOYEES WHERE DEPT_ID_DEP =

  MAX (SELECT DEPT_ID_DEP FROM DEPARTMENTS)

- SELECT * FROM EMPLOYEES WHERE DEP_ID =

  (SELECT DEPT_ID_DEP FROM DEPARTMENTS WHERE DEPT_ID_DEP IS MAX)

- SELECT * FROM EMPLOYEES WHERE DEP_ID = MAX(DEP_ID)

- SELECT * FROM EMPLOYEES WHERE DEP_ID =

  (SELECT MAX(DEPT_ID_DEP) FROM DEPARTMENTS)

Submit You have used 0 of 2 attempts Save

Question 3

1 point possible (graded)

You are writing a query that will give you the total cost to the Pet Rescue organization of rescuing animals. The cost of each rescue is stored in the Cost column. You want the result column to be called “Total_Cost”. Which of the following SQL queries is correct?

- ◯ SELECT SUM(Cost) FROM PetRescue
- ◯ SELECT SUM(Cost) AS Total_Cost FROM PetRescue
- ◯ SELECT SUM(Total_Cost) From PetRescue
- ◯ SELECT Total_Cost FROM PetRescue

Submit You have used 0 of 2 attempts Save

Module Introduction & Learning Objectives (1m)

Bookmarked

Module Introduction

In this module, you will learn the basic concepts related to using Python to connect to databases. In a Jupyter Notebook, you will create tables, load data, query data using SQL, and analyze data using Python.

Learning Objectives

- Describe concepts related to accessing Databases using Python
- Demonstrate how to perform simplified database access from Python using SQL magic
- Demonstrate writing SQL queries and retrieve result sets from Python
- Demonstrate how to connect to a database from a Jupyter notebook
- Demonstrate how to create tables and insert data from Python

How to Access Databases Using Python

Databases are powerful tools for data scientists. After completing this module,

you'll be able to explain the basic concepts related to using Python to connect to databases.

Then you'll create tables, load data and query data using SQL from Jupyter Notebooks,

and finally, analyze the data. In the lab assignments, you will learn how to create an instance in the Cloud, connect to a database, query data from the database using SQL, and analyze the data using Python.

You will be able to explain the basic concepts related to connecting a Python application to a database.

Describe SQL APIs as well as list some of the proprietary APIs used by popular SQL-based DBMS systems. Let's quickly review some of the benefits of using Python, a popular scripting language for connecting to databases.

- The Python ecosystem is very rich and provides easy to use tools for data science.

Some of the most popular packages are NumPy, pandas, matplotlib, and SciPy.

- Python is easy to learn and has a simple syntax. Due to its open source nature,

Python has been ported to many platforms. All your python programs can work on any of

these platforms without requiring any changes at all. If you are careful and avoid any system dependent features, Python supports relational database systems.

- Writing Python code to access databases is made easier by the presence of the Python database API.

Commonly referred to as the DB API, and detailed documentation related to Python is easily available.

Notebooks are also very popular in the field of data science because they run in an environment that allows creation and sharing of documents that contain live code, equations, visualizations, and explanatory texts.

A notebook interface is a virtual notebook environment used for programming. Examples of notebook interfaces include the Mathematica notebook, Maple worksheet, Matlab notebook, IPython Jupyter, R Markdown, Apache Zeppelin, Apache Spark notebook, and the Databricks cloud.

In this module, we will be using Jupyter notebooks. The Jupyter notebook is an open source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative texts.

Here are some of the advantages of using Jupyter notebooks.

- Notebook support for over 40 programming languages including Python, R, Julia, and Scala.
- Notebooks can be shared with others by email, Dropbox, GitHub, and the Jupyter notebook viewer.
- Your code can produce rich interactive output HTML, images, videos, LaTex, and customized types.
- You can leverage big data tools such as Apache Spark from Python, R, and Scala, and explore that same data with pandas, scikit-learn, ggplot2, and TensorFlow.

This is how a typical user accesses databases using Python code written on a Jupyter notebook,a web based editor.

- There is a mechanism by which the Python program communicates with the DBMS.
- The Python code connects to the database using API calls. We will explain the basics of SQL APIs and Python DB APIs.
- 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.

Now, lets learn basic concepts about some of the proprietary APIs used by popular SQL-based DBMS systems.

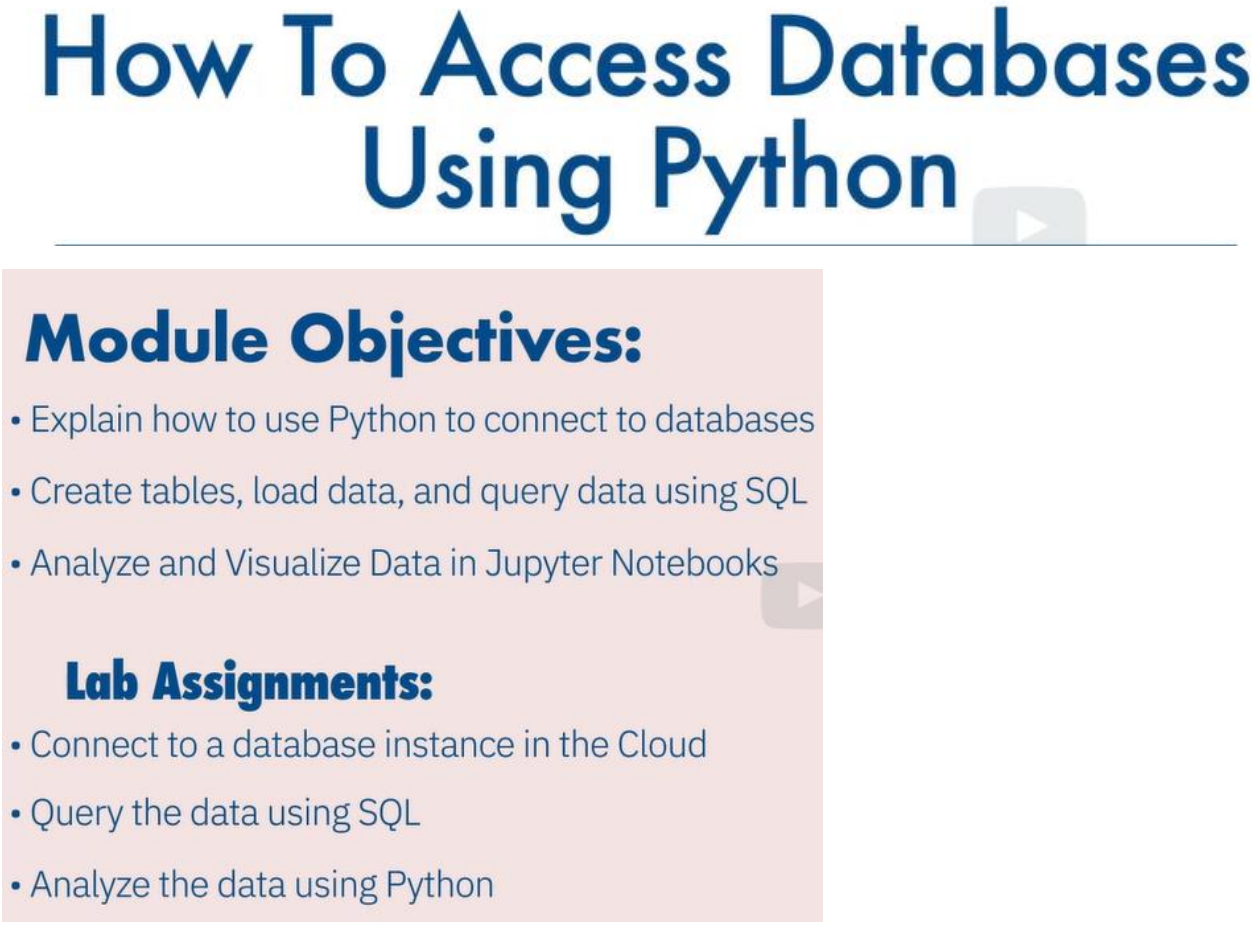
- Each database system has its own library. As you can see, the table shows a list of a few applications and corresponding SQL APIs.

- MySQL C API provides low level access to the MySQL client server protocol and enables C programs to access database contents.

- The psycopg2 API connects Python applications in PostgreSQL databases. The IBM_DB API is used to connect Python applications to IBM DB2 databases. The dblib API is used to connect to SQL server databases.

ODBC is used for database access for Microsoft Windows OS. OCI is used by Oracle databases.

And finally, JDBC is used by Java applications.

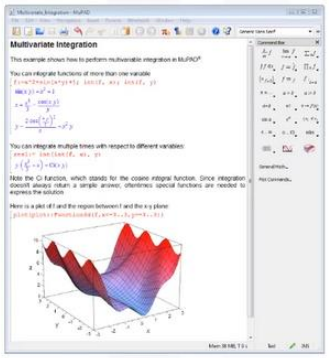


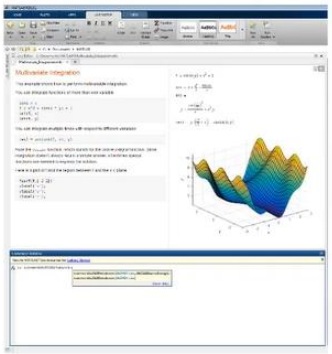
Benefits of python for database programming

- Python ecosystem : NumPy, pandas, matplotlib, SciPy
- Ease of use
- Portable
- Python supports relational database systems
- Python Database API (DB-API)

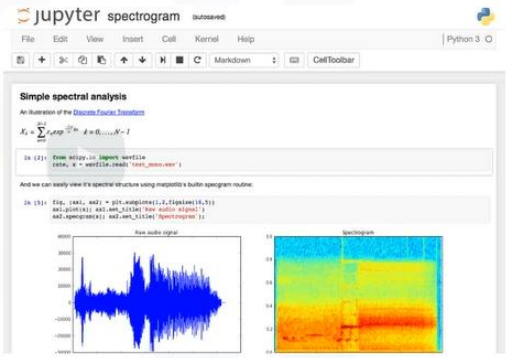
Introduction to notebooks

Matlab notebook





Jupyter notebook



What are Jupyter Notebooks?

- Language of choice
- Share notebooks
- Interactive output
- Big data integration

Accessing databases using Python

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

Hello and welcome to writing code using DB-APIs. After completing this video, you will be able to explain the basic concepts related to the Python DB-API and database cursors.

And also write code using DB-APIs. As we saw in the beginning of this module, the user writes Python programs using a Jupyter notebook.

The Python code connects to the database using DB-API calls. There is a mechanism by which the Python code communicates with the DBMS.

DB-API is

- Python's standard API for accessing relational databases.
- It is a standard that allows you to write a single program that works with multiple kinds of relational databases instead of writing a separate program for each one.

So, if you learn the DB-API functions, then you can apply that knowledge to use any database

with Python.

Here are some Advantages of using the DB-API.

- It's easy to implement and understand.
- This API has been defined to encourage similarity between the Python modules that are used to access databases.
- It achieves consistency which leads to more easily understood modules. The code is generally more portable across databases, and it has a broader reach of database connectivity from Python. As we know, each database system has its own library.

As you can see, the table shows a list of a few databases and corresponding DB-APIs to connect to Python applications.

The IBM_db library is used to connect to an IBM DB2 database.

The MySQL Connector/Python library is used to connect to a Compose for MySQL database.

The psycopg2 library is used to connect to a Compose from PostgreSQL database.

And finally, the PyMongo library is used to connect to a Compose for MongoDB database.

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.

Let's walk through a Python application that uses the DB-API to query a database.

2. First, you import your database module by using the connect API from that module.
3. 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.
4. The connect function returns connection object. After this, you create a cursor object on the connection object.
5. 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.

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

Concepts of the Python DB API

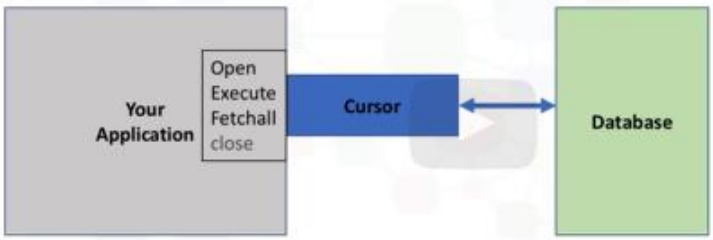
Connection Objects
- Database connections
- Manage transactions

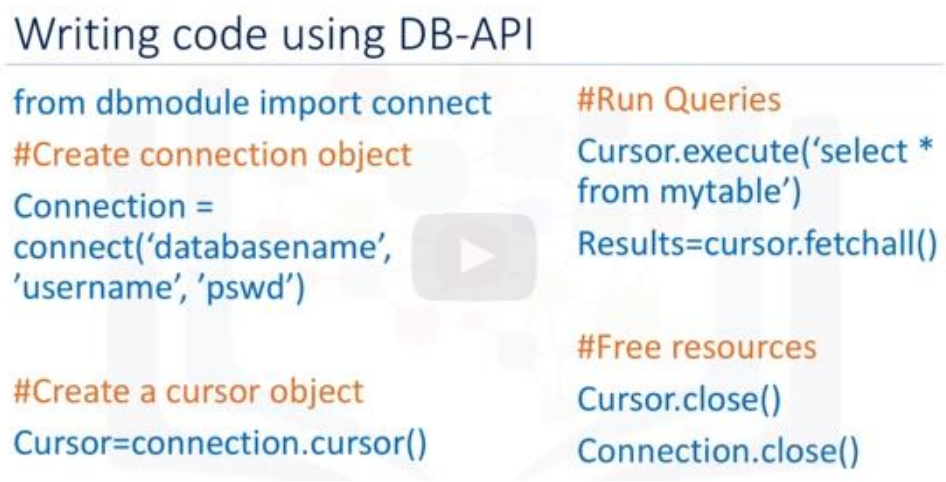
Cursor Objects
- Database Queries
- Scroll through result set
- Retrieve results

What are cursor methods?

- .callproc()
- .execute()
- .executemany()
- .fetchone()
- .fetchmany()
- .fetchall()
- .nextset()
- .arraysize()
- .close()

What is a database cursor?





Connecting to a Database Using ibm_db API

After completing this lesson, you will be able to understand the ibm_db API,

as well as the credentials required to connect to a database using Python.

We will also demonstrate how to connect to an IBM DB2 database using Python code written on a Jupyter notebook.

The ibm_db API

- provides a variety of useful Python functions for accessing and manipulating data in an IBM data server database, including functions for connecting to a database,

preparing and issuing SQL statements, fetching rows from result sets, calling stored procedures, committing and rolling back transactions, handling errors and retrieving metadata.

- The ibm_db API uses the IBM Data Server Driver for ODBC, and CLI APIs to connect to IBM, DB2, and Informix.

We import the ibm_db library into our Python application.

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.

Here is an example of creating a DB2 database connection in Python.

We create a connection object DSN, which stores the connection credentials.

The connect function of the ibm_db API will be used to create a non persistent connection.

The DSN object is passed as a parameter to the connection function. If a connection has been established with the database, then the code returns connected, as the output otherwise, the output will be unable to connect to database. Then we free all resources by closing the connection. Remember that it is always important to close connections so that we can avoid unused connections taking up resources.

Connecting to a database using ibm_db API

What is ibm_db?

- The ibm_db API provides a variety of useful Python functions for accessing and manipulating data in an IBM data server Database
- Ibm_db API uses the IBM Data Server Driver for ODBC and CLI APIs to connect to IBM DB2 and Informix

Identify database connection credentials

```
dsn_driver = "{IBM DB2 ODBC DRIVER}"
dsn_database = "BLUDB"            # e.g. "BLUDB"
dsn_hostname = "YourDb2Hostname" # e.g.: "dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net"
dsn_port = "50000"                # e.g. "50000"
dsn_protocol = "TCPIP"            # i.e. "TCPIP"
dsn_uid = "***************"        # e.g. "abc12345"
dsn_pwd = "***************"      # e.g. "7dBZ3wWt9XN6$o0J"
```

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!
```

Create credentials to access your database instance

Objective(s)
At the end of this lab you will be able to:

- Create your own instance of database on IBM Cloud using DB2 service

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

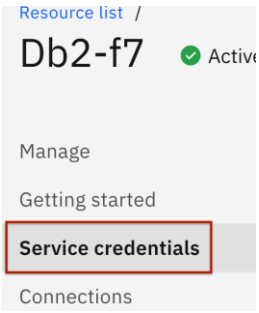
Note: you may need to log into IBM Cloud in the process of loading the resources/dashboard.

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

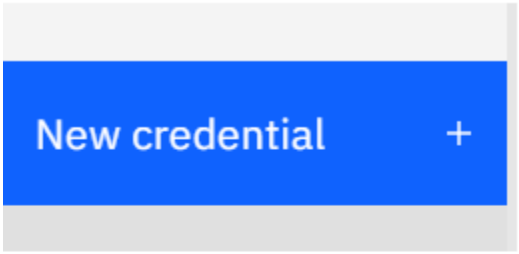
1. Locate and click on your Db2 service listed under Services.

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

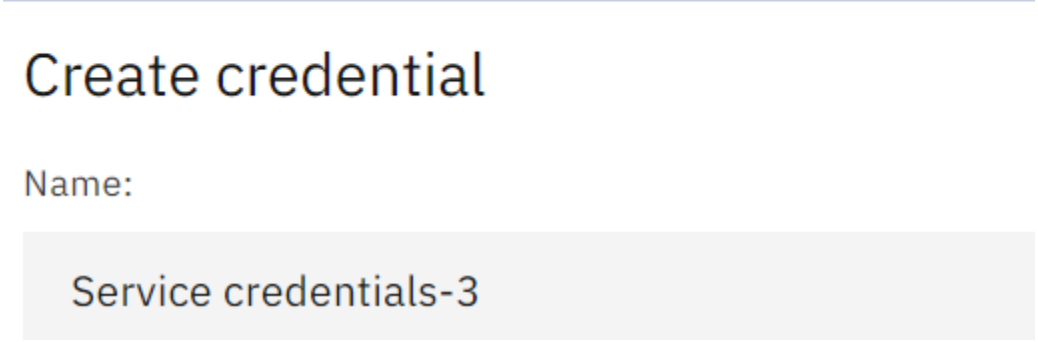
2. Click on Service Credentials in the left menu

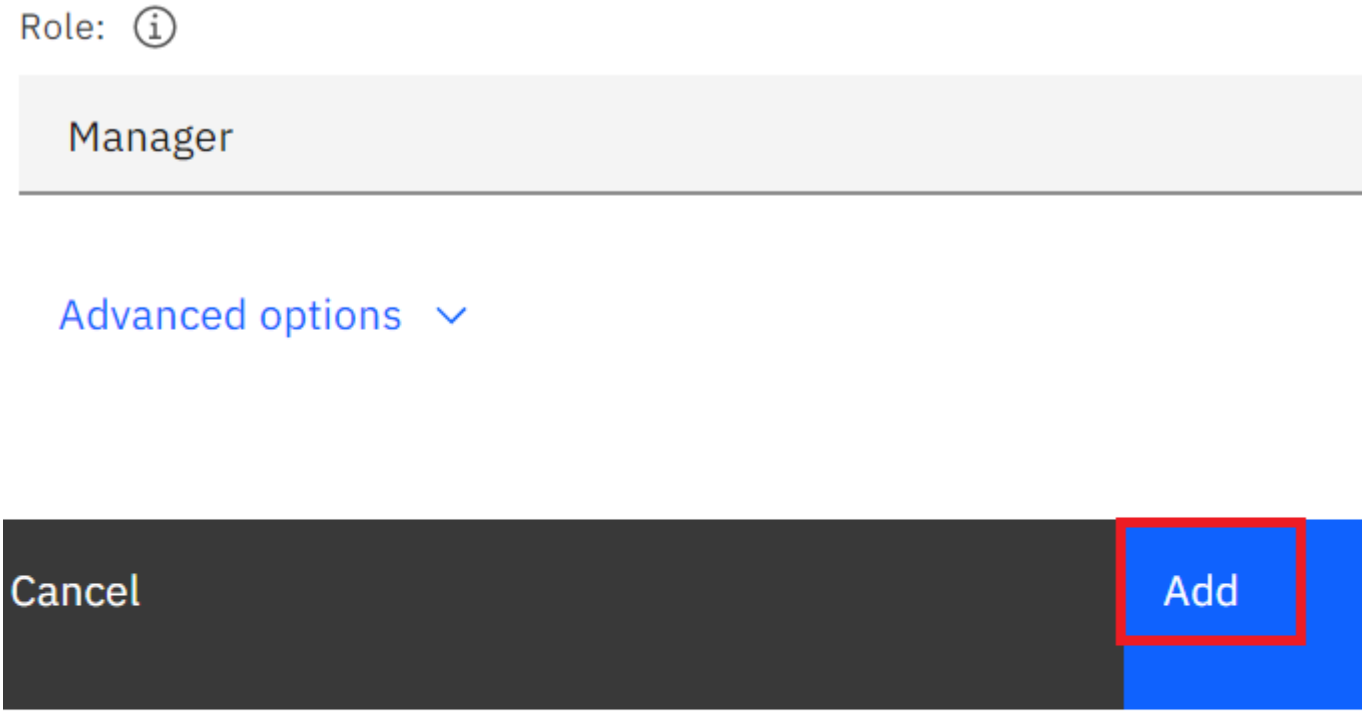


3. Click on the button to create **New credential**



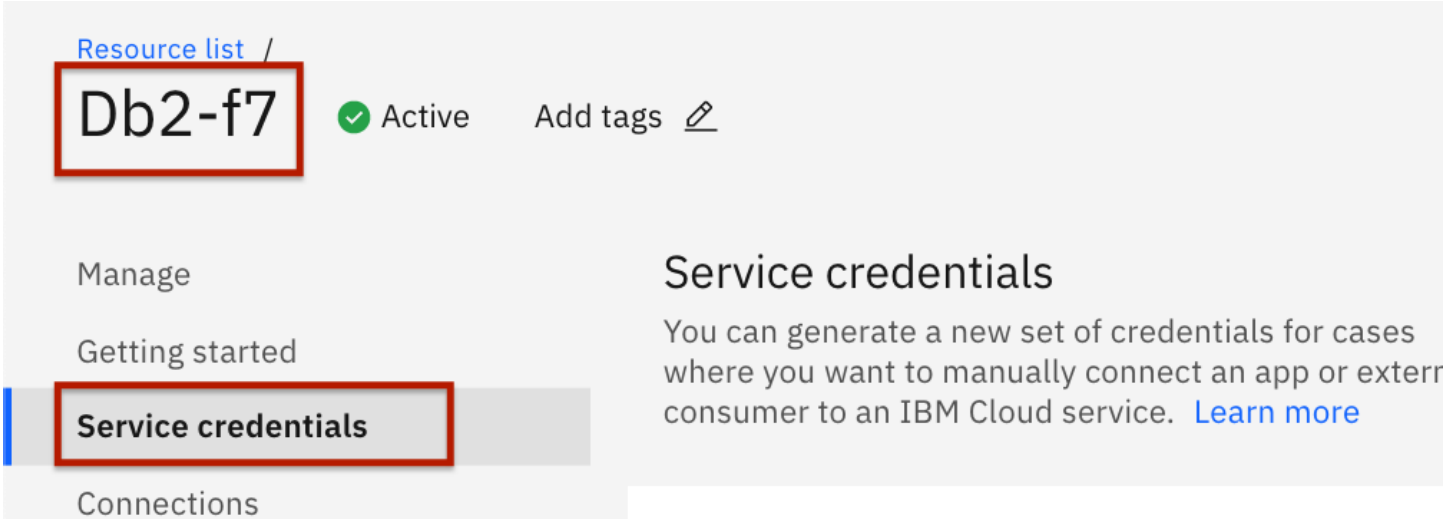
n the prompt that comes up click the “Add” button in the bottom right:

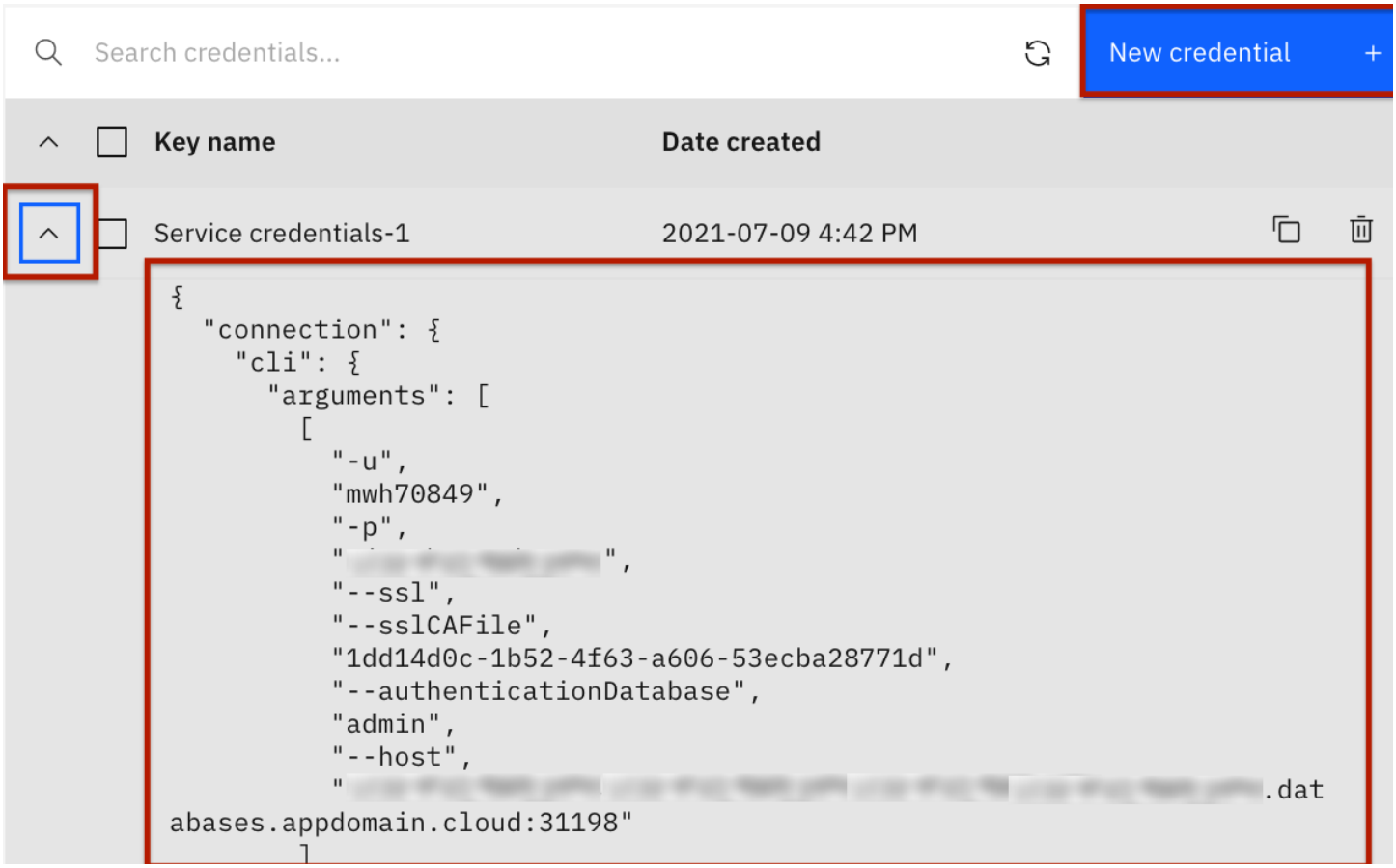




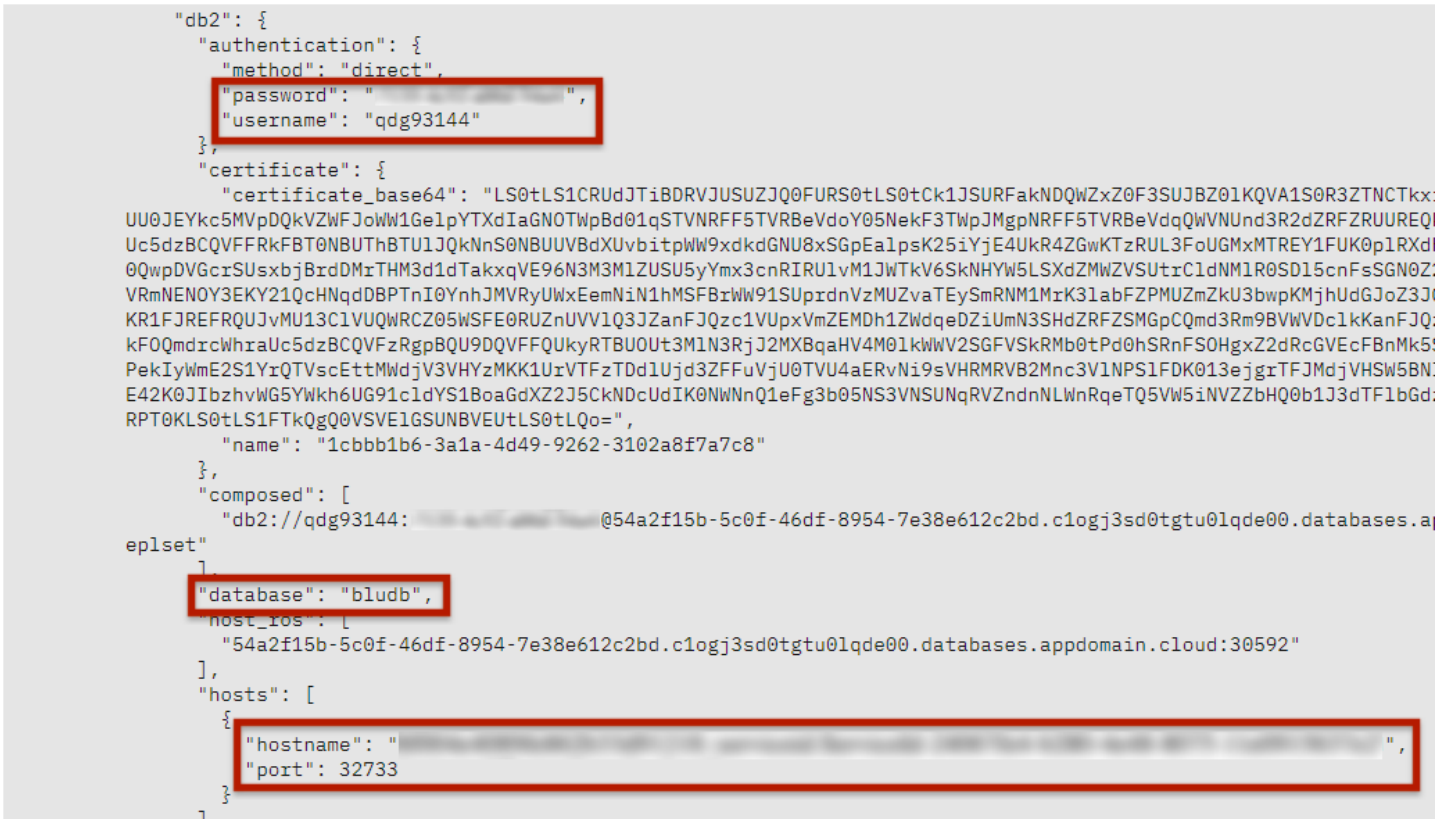
4. Check the box to **View credentials**
5. Copy and save the credentials making a note of the following:

- **port** is the database port
- **db** is the database name
- **host** is the hostname of the database instance
- **username** is the username you'll use to connect
- **password** is the password you'll use to connect





You need to scroll down to get the credentials details.



```
{
  "connection": {
    "cli": {
      "arguments": [
        [
          "-u",
          "pgh93728",
          "-p",
          "rstLVzsbrLlLZEzG",
          "--ssl",
          "--sslCAFile",
          "1dd14d0c-1b52-4f63-a606-53ecba28771d",
          "--authenticationDatabase",
          "admin",
          "--host",
          "55fbc997-9266-4331-afd3-
888b05e734c0.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31929"
        ]
"password": "rstLVzsbrLlLZEzG",
        "username": "pgh93728"
"database": "bludb"
"hostname": "55fbc997-9266-4331-afd3-
888b05e734c0.bs2io90l08kqb1od8lcg.databases.appdomain.cloud",
          "port": 31929
```

Connect to Db2 database on Cloud using Python

Estimated time needed: **15** minutes

Objectives

After completing this lab you will be able to:

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

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

Import the `ibm_db` Python library

The `ibm_db` API provides a variety of useful Python functions for accessing and manipulating data in an IBM® data server database, including functions for connecting to a database, preparing and issuing SQL statements, fetching rows from result sets, calling stored procedures, committing and rolling back transactions, handling errors, and retrieving metadata.

We first import the ibm_db library into our Python Application

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

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

[ ]:
```
# These libraries are pre-installed in SN Labs. If running in another environment please uncomment lines below to install them:
# !pip install --force-reinstall ibm_db==3.1.0 ibm_db_sa==0.3.3
# Ensure we don't load_ext with sqlalchemy>=1.4 (incompadible)
# !pip uninstall sqlalchemy==1.4 -y && pip install sqlalchemy==1.3.24
# !pip install ipython-sql
```

[ ]:
```
import ibm_db
```

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

Identify the database connection credentials

Connecting to dashDB or DB2 database requires the following information:

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

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

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

```
#Replace the placeholder values with your actual Db2 hostname, username, and password:
dsn_hostname = "YourDb2Hostname" # e.g.: "54a2f15b-5c0f-46df-8954-7e38e612c2bd.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud"
dsn_uid = "YourDb2Username"        # e.g. "abc12345"
dsn_pwd = "YoueDb2Password"      # e.g. "7dBZ3wWt9XN6$o0J"

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

Create the DB2 database connection

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

Lets build the dsn connection string using the credentials you entered above

Now establish the connection to the database

```
[ ]: #DO NOT MODIFY THIS CELL. Just RUN it with Shift + Enter
#Create database connection

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

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

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

```
[ ]: #Retrieve Metadata for the Database Server
server = ibm_db.server_info(conn)

print ("DBMS_NAME: ", server.DBMS_NAME)
print ("DBMS_VER:  ", server.DBMS_VER)
print ("DB_NAME:   ", server.DB_NAME)
```

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

print ("DRIVER_NAME:          ", client.DRIVER_NAME)
print ("DRIVER_VER:           ", client.DRIVER_VER)
print ("DATA_SOURCE_NAME:     ", client.DATA_SOURCE_NAME)
print ("DRIVER_ODBC_VER:      ", client.DRIVER_ODBC_VER)
print ("ODBC_VER:             ", client.ODBC_VER)
print ("ODBC_SQL_CONFORMANCE: ", client.ODBC_SQL_CONFORMANCE)
print ("APPL_CODEPAGE:        ", client.APPL_CODEPAGE)
print ("CONN_CODEPAGE:        ", client.CONN_CODEPAGE)
```

Close the Connection

We free all resources by closing the connection. Remember that it is always important to close connections so that we can avoid unused connections taking up resources.

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

Summary

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

Creating Tables, Loading Data and Querying Data

Hello, and welcome to creating tables, loading data, and querying data.

After completing this lesson, you will be able to understand basic concepts related to creating tables, loading data, and querying data using Python, as well as demonstrate an example of how to perform these tasks using the IBM DB2 on Cloud database and Jupyter notebooks.

For this example, we will be using DB2 as the database. We first obtain a connection resource by connecting to the database by using the connect method of the ibm_db api.

There are different ways of creating tables in DB2.

1. One is using the Web console provided by DB2, and the other option is to create tables from any SQL, R, or Python environments.

Let's take a look at how to create tables in DB2 from our Python application.

Here is a sample table of a commercial Trucks 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.

Here is the code to create a table called Trucks in Python. We use the ibm_db.exec_immediate function of the ibm_db api. The connection resource that was created is passed as the first parameter to this function. The next parameter is the SQL statement, which is the create table query used to create the Trucks table. The new table created will have five columns, serial_no will be the primary key. Now let's take a look at loading data. We use the ibm_db.exec_immediate function of the ibm_db api.

The connection resource that was created is passed as the first parameter to this function.

The next parameter is the SQL statement, which is the insert into query used to insert data in the Trucks table.

A new row will be added to the Trucks table. Similarly, we add more rows to the Trucks table using the ibm_db.exec_immediate function. Now that your Python code has been connected to a database instance and the database table has been created and populated with data, let's see how we can fetch data from the Trucks table that we created on DB2 using Python code. We use the ibm_db.exec_immediate function of the ibm_db api.

The connection resource that was created is passed as the first parameter to this function.

The next parameter is the SQL statement, which is the select from table query.

The Python code returns the output, which shows the fields of the data in the Trucks table.

You can check if the output returned by the select query shown is correct, by referring to the DB2 console.

Let's look at how we can use pandas to retrieve data from the database tables.

Pandas is a popular Python library that contains high level data structures and manipulation tools designed to make data analysis fast and easy in Python.

We load data from the Trucks table into a data frame called DF. A data frame represents a tabular spreadsheet like data structure containing an ordered collection of columns, each of which can be a different value type.

Creating tables

Serial No	Model	Manufacturer	Engine Size	Class
A1234	Lonestar	International Trucks	Cummins ISX15	Class 8
B5432	Volvo VN	Volvo Trucks	Volvo D11	Heavy Duty Tractor Class 8
C5674	Kenworth W900	Kenworth Truck Company	Caterpillar C9	Class 8

ibm_db.exec_immediate()

The parameters for the function are:

- Connection
- Statement
- Options

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'}
```

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

Introducing SQL Magic

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.

In the following tutorial you will learn how to work with SQL magic.

Analyzing Data with Python

Hello, and welcome to analyzing data with Python. After completing this video,

you will be able to understand basic concepts related to performing exploratory analysis on data.

We will demonstrate an example of how to store data using the IBM Db2 on Cloud database,

and then use Python to do some basic data analysis on this data. In this video, we will be using

the McDonald's menu nutritional facts data for popular menu items at McDonald's, while using Python to perform basic exploratory analysis.

McDonald's is an American fast food company and the world's largest restaurant chain by revenue.

Although McDonald's is known for fast food items such as hamburgers, French fries, soft drinks, milkshakes, and desserts, the company has added to its menu salads, fish, smoothies, and fruit.

McDonald's provides nutrition analysis of their menu items to help you balance your McDonald's meal with other foods you eat.

The data set used in this lesson has been obtained from the nutritional facts for McDonald's menu from Kaggle. We need to create a table on Db2 to store

the McDonald's menu nutrition facts data set that we will be using. We will also be using the console provided by Db2 for this process.

There are four steps involved in loading data into a table, source, target, define, and finalize.

We first load the spreadsheet into the Db2 using the console. We then select the target schema,

and then you will be given an option to load the data into an existing table or create a new table.

When you choose to create a new table, you have the option to specify the table name.

Next, you will see a preview of the data where you can also define the columns and data types.

Review the settings and begin the load. When the loading is complete,

you can see the statistics on the loaded data. Next, view the table to explore further.

Db2 Warehouse allows you to analyze data using in-database analytics, APIs, RStudio or Python.

The data has been loaded into our relational database. You can run Python scripts that retrieve data from and write data to a Db2 database.

Such scripts can be powerful tools to help you analyze your data. For example, you can use them to generate statistical models based on data in your database,

and to plot the results of these models. In this lesson, we will be using Python scripts that will be run within a Jupyter notebook.

Now, after obtaining a connection resource, by connecting to the database, by using the connect method of the IBM_DB API,

we use the SQL select query to verify the number of rows that have been loaded in the table created.

The figure shows a snapshot of the output. The output obtained is 260 which is similar

to the number of rows in the Db2 console. Now let's see how we can use Pandas to retrieve data from the database tables.

We load data from the McDonalds_nutrition table into the data frame DF using the read_SQL method.

The SQL select query and the connection object are passed as parameters to the read_SQL method.

We can view the first few rows of the data frame DF that we created using the head method.

Now it's time to learn about your data. Pandas methods are equipped with a set of common mathematical and statistical methods.

Let's use the describe method to view the summary statistics of the data in the data frame,

then explore the output of the describe method. We see that there are 260 observations or food items in our data frame.

We also see that there are nine unique categories of food items in our data frame. We can also see summary statistics information such as frequency, mean, median, standard deviation, et cetera for the 260 food items across the different variables.

For example, the maximum value for total fat is 118.

Let's investigate this data further. Let's try to understand one of the nutrients in the food items which is sodium. A main source of sodium is table salt.

The average American eats five or more teaspoons of salt each day. This is about 20 times as much as the body needs. Sodium is found naturally in foods, but a lot of it is added during processing and preparation. Many foods that do not taste salty, may still be high in sodium.

Large amounts of sodium can be hidden in canned, processed and convenience foods.

Sodium controls fluid balance in our bodies, and maintains blood volume and blood pressure.

Eating too much sodium may raise blood pressure and cause fluid retention, which could lead to swelling of the legs, and feet, or other health issues.

When limiting sodium in your diet, a common target is to eat less than 2,000 milligrams of sodium per day.

Now using the nutrition data set for McDonald's, let's do some basic data analysis to answer the question.

Which food item has the maximum sodium content? We first use visualization to explore the sodium content of food items. Using the swarm plot method provided by the Seaborne package,

we create a categorical scatter plot as shown on the right, then give as the input, category on the x-axis, sodium on the y-axis, and the data will be the data frame DF that contains the nutritional data set from McDonald's.

The plot shows the sodium values for the different food items by category. Notice a high value of around 3,600 for sodium on the scatter plot. We will be learning about visualizations later in this module.

Let's further explore this high sodium value and identify which food items on the menu have this value for sodium.

Let's do some basic data analysis using Python to find which food items on the menu have maximum sodium content.

To check the values of sodium levels in the food items within the dataset, we use the code as shown in code 1.

The describe method is used to understand the summary statistics associated with sodium.

Notice that the maximum value of sodium is given as 3,600. Now let's further explore the row associated with the maximum sodium variable as shown in code 2.

We use the idxmax method to compute the index values, at which the maximum value of sodium is obtained in the data frame. We see that the output is 82.

Now lets find the item name associated with the 82nd item in our data frame. As shown in code 3, we will use the .at method to find the item name by passing the index of 82 and the column name item, to be returned for the 82nd row.

Finally, we find that the food item on the menu that has a highest sodium content is Chicken McNuggets, 40 pieces. Visualizations are very useful for initial data exploration.

They can help us understand relationships, patterns, and outliers in the data.

Let's first create a scatter plot with protein on the x-axis, and total fat on the y-axis.

Scatter plots are very popular visualization tools and show the relationship between two variables with a point for each observation.

To do this, we can use the joint plot function provided by the Seaborn package, and give as input, protein on the x-axis and total fat on the y-axis. And the data will be the data frame DF that contains the nutritional data set from McDonald's.

The output scatter plot is shown on the right side. The plot has an interesting shape.

It shows the correlation between the two variables: protein and fat. Correlation is a measure of association between two variables, and has a value of between -1 and +1.

We see that the points on the scatter plot are closer to a straight line in the positive direction.

So we have a positive correlation between the two variables. On the top right corner of the scatter plot, we have the values of the Pearson correlation- 0.81 and the significance of the correlation denoted as P - which is a good value that shows the variables are certainly correlated.

The plot also shows two histograms: one on the top and the other on the right side.

The histogram on the top is that of the variable protein, and the histogram on the right side is that of the variable total fat.

We also noticed that there is a point on the scatter plot outside the general pattern.

This is a possible outlier. Now let's see how we can visualize data using box plots. Box plots are charts that indicate the distribution of one or more variables.

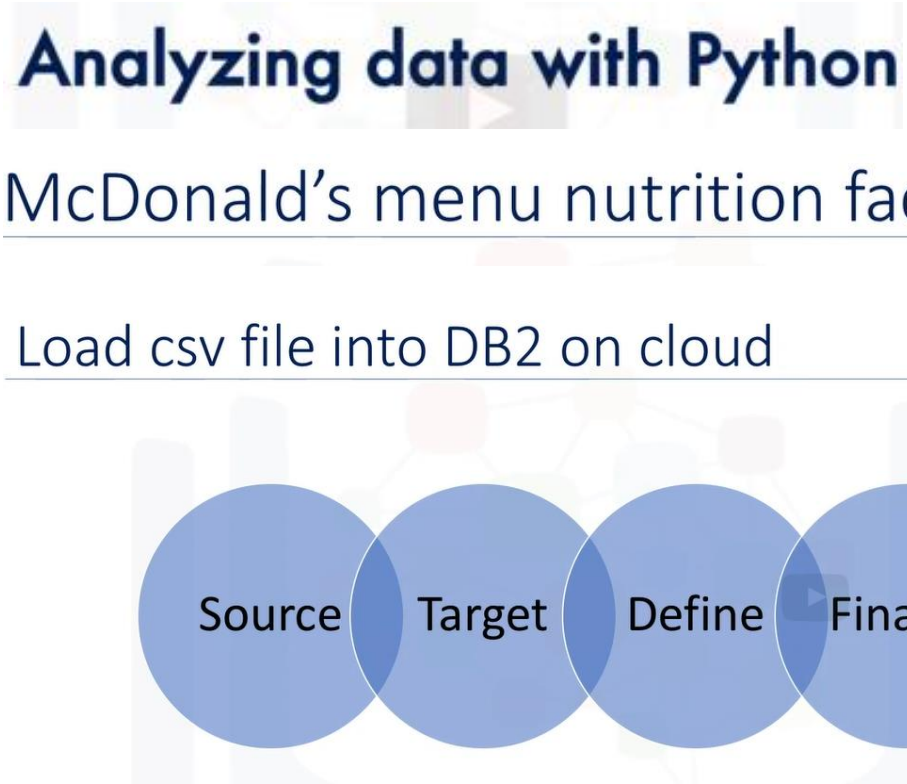
The box in a box plot captures the middle 50 percent of data. Lines and points indicate possible skewness and outliers. Let's create a box plot for sugar.

The function we are going to use is box plot from the Seaborn package. We give the column name sugars as input to the box plot function.

The output is shown on the right side, where we had the box plot with average values of sugar and food items around 30 grams. We also notice a few outliers that indicate food items with extreme values of sugar. There exist food items in the data set that have sugar content of around 128 grams.

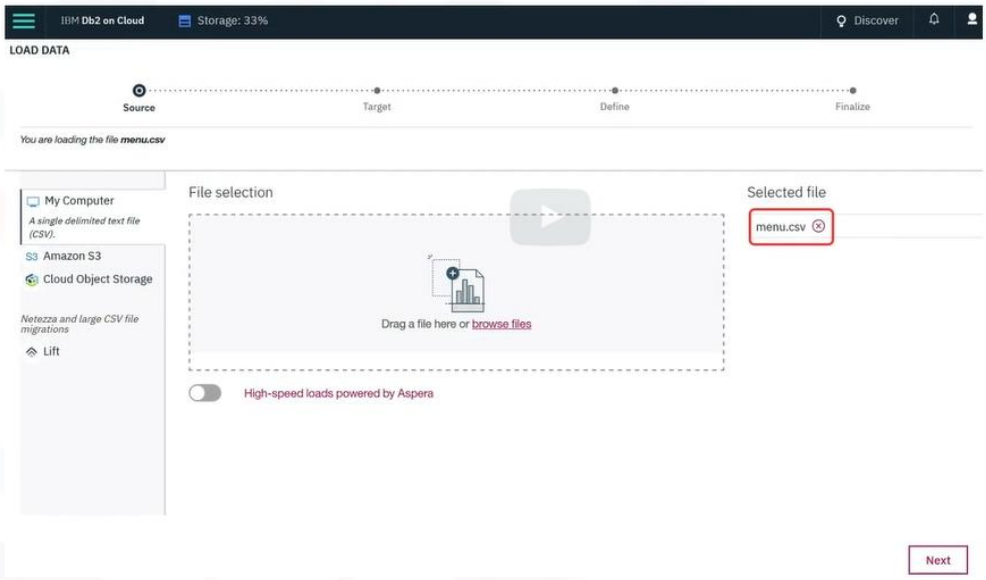
Candies maybe among these high sugar content food items on the menu. Now that you know how to do basic exploratory data analysis using Pandas and visualization tools,

proceed to the labs in this module where you can practice the concepts learned.



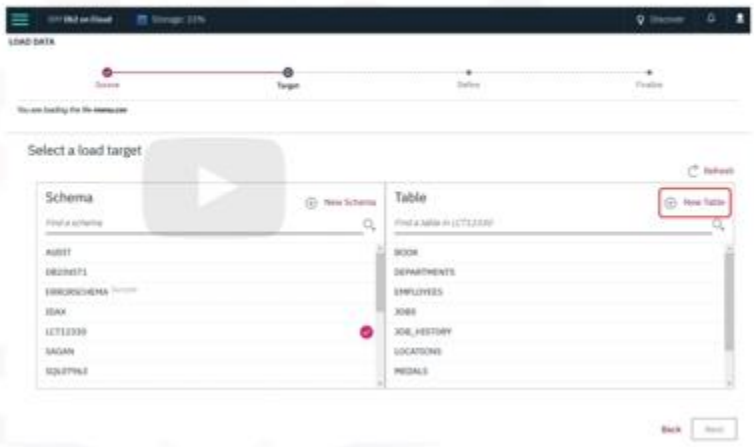
Load csv file into DB2 on cloud

- Source



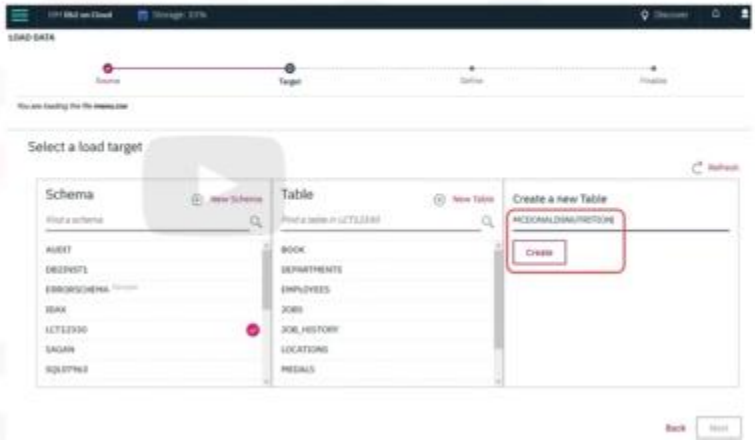
Load csv file into DB2 on cloud

- Source
- Target



Load csv file into DB2 on cloud

- Source
- Target
- Define

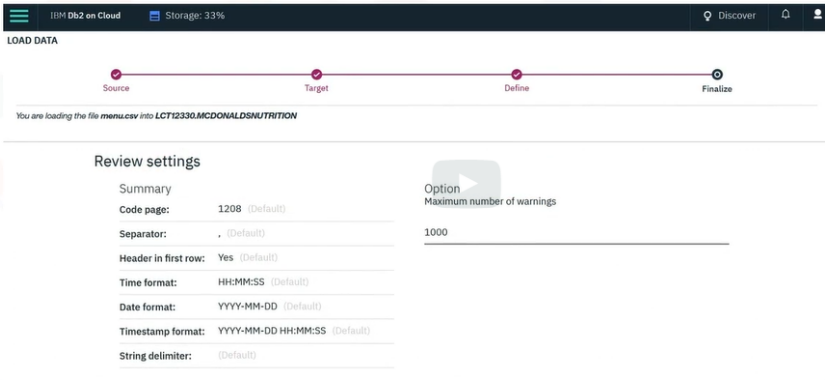


Load csv file into DB2 on cloud

- Source
- Target
- Define

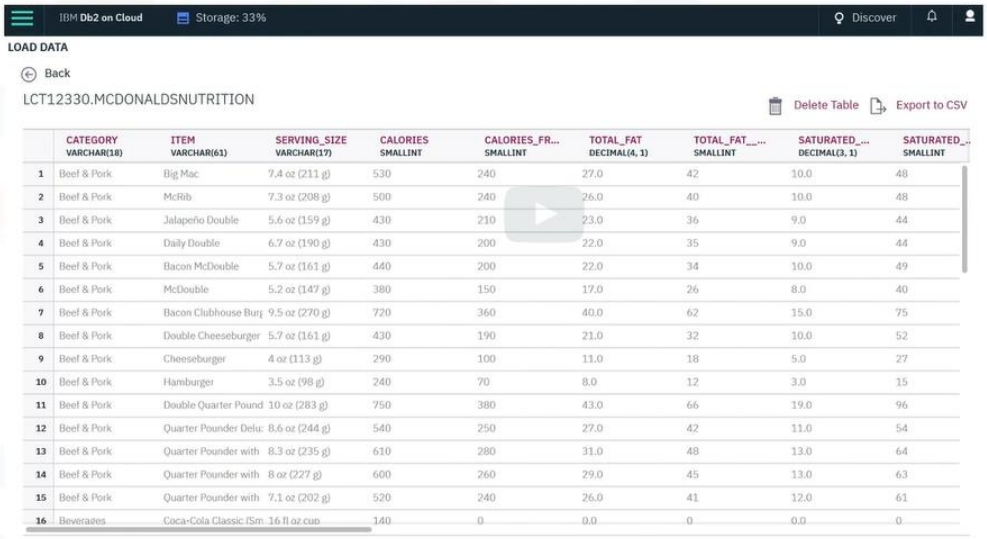
Load csv file into DB2 on cloud

- Source
- Target
- Define
- Finalize



Load csv file into DB2 on cloud

- Source
- Target
- Define
- Finalize



	CATEGORY VARCHAR(18)	ITEM VARCHAR(61)	SERVING_SIZE VARCHAR(17)	CALORIES SMALLINT	CALORIES_FR... SMALLINT	TOTAL_FAT DECIMAL(4, 1)	TOTAL_FAT_... SMALLINT	SATURATED_... DECIMAL(3, 1)	SATURATED_... SMALLINT
1	Beef & Pork	Big Mac	7.4 oz (211 g)	530	240	27.0	42	10.0	48
2	Beef & Pork	McRib	7.3 oz (208 g)	500	240	26.0	40	10.0	48
3	Beef & Pork	Jalapeño Double	5.6 oz (159 g)	430	210	23.0	36	9.0	44
4	Beef & Pork	Daily Double	6.7 oz (190 g)	430	200	22.0	35	9.0	44
5	Beef & Pork	Bacon McDouble	5.7 oz (161 g)	440	200	22.0	34	10.0	49
6	Beef & Pork	McDouble	5.2 oz (147 g)	380	150	17.0	26	8.0	40
7	Beef & Pork	Bacon Clubhouse Burg	9.5 oz (270 g)	720	360	40.0	62	15.0	75
8	Beef & Pork	Double Cheeseburger	5.7 oz (161 g)	430	190	21.0	32	10.0	52
9	Beef & Pork	Cheeseburger	4 oz (113 g)	290	100	11.0	18	5.0	27
10	Beef & Pork	Hamburger	3.5 oz (98 g)	240	70	8.0	12	3.0	15
11	Beef & Pork	Double Quarter Pound	10 oz (283 g)	750	380	43.0	66	19.0	96
12	Beef & Pork	Quarter Pounder Delu	8.6 oz (244 g)	540	250	27.0	42	11.0	54
13	Beef & Pork	Quarter Pounder with	8.3 oz (235 g)	610	280	31.0	48	13.0	64
14	Beef & Pork	Quarter Pounder with	8 oz (227 g)	600	260	29.0	45	13.0	63
15	Beef & Pork	Quarter Pounder with	7.1 oz (202 g)	520	240	26.0	41	12.0	61
16	Beverages	Coca-Cola Classic (Sm	16 fl oz cup	140	0	0.0	0	0.0	0

Verify Loaded Data Using SQL

Mc Donalds nurtition

```
In [ ]: ### Verify Loaded Data Using SQL
```

```
In [7]: stmt = ibm_db.exec_immediate(conn,"SELECT count(*) FROM MCDONALDS_NUTRITION")

        ibm_db.fetch_both(stmt)

Out[7]: {0: '260', '1': '260'}
```

Using pandas

Exploratory analysis using pandas

In [5]:
```
import pandas
import ibm_db_dbi
pconn = ibm_db_dbi.Connection(conn)
df = pandas.read_sql('SELECT * FROM MCDONALDS_NUTRITION', pconn)
df
```

Out[21]:

	Category	Item	Serving Size	Calories	Calories from Fat	Total Fat	Total Fat (% Daily Value)	Saturated Fat	Saturated Fat (% Daily Value)	Trans Fat	...	Carbohydrates	Carbohydrates (% Daily Value)	Dietary Fiber	Di Fit (% Da Va
0	Breakfast	Egg McMuffin	4.8 oz (136 g)	300	120	13.0	20	5.0	25	0.0	...	31	10	4	17
1	Breakfast	Egg White Delight	4.8 oz (135 g)	250	70	8.0	12	3.0	15	0.0	...	30	10	4	17
2	Breakfast	Sausage McMuffin	3.9 oz (111 g)	370	200	23.0	35	8.0	42	0.0	...	29	10	4	17
3	Breakfast	Sausage McMuffin with Egg	5.7 oz (161 g)	450	250	28.0	43	10.0	52	0.0	...	30	10	4	17
4	Breakfast	Sausage McMuffin with Egg Whites	5.7 oz (161 g)	400	210	23.0	35	8.0	42	0.0	...	30	10	4	17
5	Breakfast	Steak & Egg McMuffin	6.5 oz (185 g)	430	210	23.0	36	9.0	46	1.0	...	31	10	4	18

Learn about your data

In [34]:
```
df.describe(include='all')
```

Out[34]:

	Category	Item	Serving Size	Calories	Calories from Fat	Total Fat	Total Fat (% Daily Value)	Saturated Fat	Saturated Fat (% Daily Value)	Trans Fat	...	Carbohydrates	Ca (% Va
count	260	260	260	260.000000	260.000000	260.000000	260.000000	260.000000	260.000000	260.000000	...	260.000000	26
unique	9	260	107	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na
top	Coffee & Tea	Nonfat Caramel Mocha (Large)	16 fl oz cup	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na
freq	95	1	45	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na
mean	NaN	NaN	NaN	368.269231	127.096154	14.165385	21.815385	6.007692	29.965385	0.203846	...	47.346154	15
std	NaN	NaN	NaN	240.269886	127.875914	14.205998	21.885199	5.321873	26.639209	0.429133	...	28.252232	9.4
min	NaN	NaN	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.(
25%	NaN	NaN	NaN	210.000000	20.000000	2.375000	3.750000	1.000000	4.750000	0.000000	...	30.000000	10
50%	NaN	NaN	NaN	340.000000	100.000000	11.000000	17.000000	5.000000	24.000000	0.000000	...	44.000000	15
75%	NaN	NaN	NaN	500.000000	200.000000	22.250000	35.000000	10.000000	48.000000	0.000000	...	60.000000	20
max	NaN	NaN	NaN	1880.000000	1060.000000	118.000000	182.000000	20.000000	102.000000	2.500000	...	141.000000	47

Which food item has maximum sodium content?

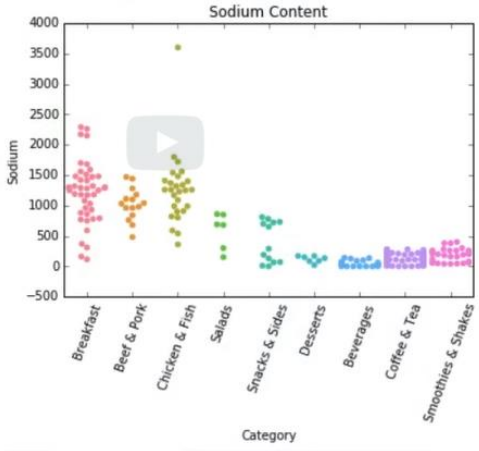
- Main source of sodium is table salt
- Average American eats 5 teaspoons/day
- Sodium mostly added during preparation
- Foods that don't taste salty may be high in sodium
- Sodium controls fluid balance in our bodies
- Too much sodium may raise blood pressure
- Target less than 2,000 milligrams/day

Which food item has maximum sodium content?

```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

```
### Categorical scatterplots
```

```
plot = sns.swarmplot(x="Category", y='Sodium', data=df)
plt.setp(plot.get_xticklabels(), rotation=70)
plt.title('Sodium Content')
plt.show()
```

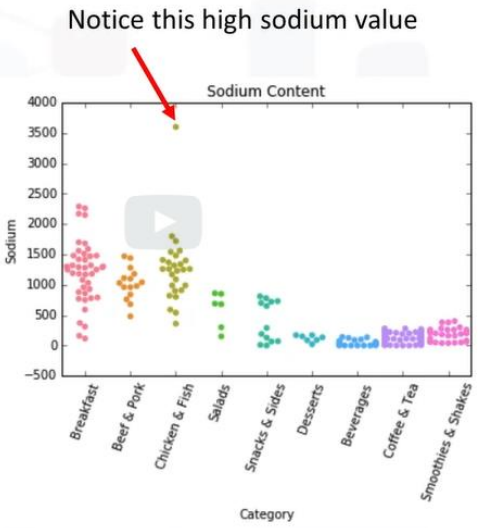


Which food item has maximum sodium content?

```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

```
### Categorical scatterplots
```

```
plot = sns.swarmplot(x="Category", y='Sodium', data=df)
plt.setp(plot.get_xticklabels(), rotation=70)
plt.title('Sodium Content')
plt.show()
```



Which food item has maximum sodium content?

Code 1

```
In [17]: df['Sodium'].describe()

Out[17]: count     260.000000
         mean      495.750000
         std       577.026323
         min         0.000000
         25%       107.500000
         50%       190.000000
         75%       865.000000
         max      3600.000000
         Name: Sodium, dtype: float64
```

Which food item has maximum sodium content?

Code 1

```
In [17]: df['Sodium'].describe()

Out[17]: count     260.000000
         mean      495.750000
         std       577.026323
         min         0.000000
         25%       107.500000
         50%       190.000000
         75%       865.000000
         max      3600.000000
         Name: Sodium, dtype: float64
```

Code 2

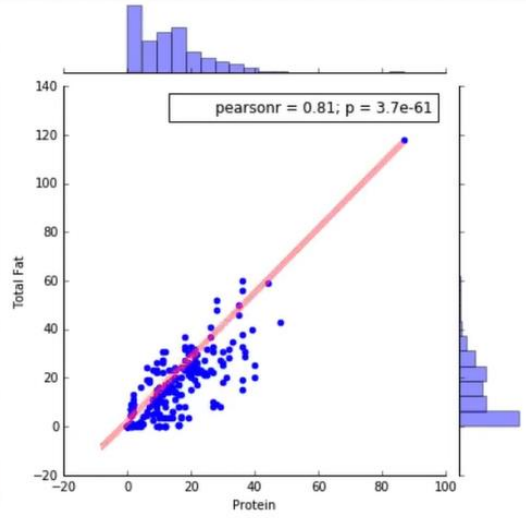
```
In [24]: df['Sodium'].idxmax()

Out[24]: 82
```

Code 3

```
In [56]: df.at[82,'Item']

Out[56]: 'Chicken McNuggets (40 piece)'
```

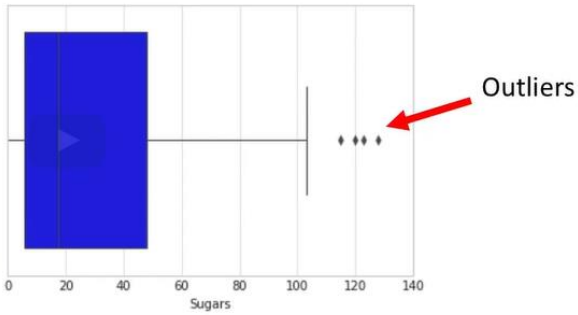
Further data exploration using visualizations

```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

plot = sns.jointplot(x="Protein", y='Total Fat', data=df)
plot.show()
```



Further data exploration using visualizations

```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

plot = sns.set_style("whitegrid")
ax = sns.boxplot(x=df["Sugars"])
plot.show()
```



Guide for the exam

IBM Developer
SKILLS NETWORK

Analyzing a real world data-set with SQL and Python

Estimated time needed: **15** minutes

Objectives

After completing this lab you will be able to:

- Understand a dataset of selected socioeconomic indicators in Chicago
- Learn how to store data in an Db2 database on IBM Cloud instance
- Solve example problems to practice your SQL skills

Selected Socioeconomic Indicators in Chicago

The city of Chicago released a dataset of socioeconomic data to the Chicago City Portal. This dataset contains a selection of six socioeconomic indicators of public health significance and a "hardship index," for each Chicago community area, for the years 2008 – 2012.

Scores on the hardship index can range from 1 to 100, with a higher index number representing a greater level of hardship.

A detailed description of the dataset can be found on the city of Chicago's website, but to summarize, the dataset has the following variables:

- **Community Area Number** (`ca`): Used to uniquely identify each row of the dataset
- **Community Area Name** (`community_area_name`): The name of the region in the city of Chicago
- **Percent of Housing Crowded** (`percent_of_housing_crowded`): Percent of occupied housing units with more than one person per room
- **Percent Households Below Poverty** (`percent_households_below_poverty`): Percent of households living below the federal poverty line
- **Percent Aged 16+ Unemployed** (`percent_aged_16_unemployed`): Percent of persons over the age of 16 years that are unemployed
- **Percent Aged 25+ without High School Diploma** (`percent_aged_25_without_high_school_diploma`): Percent of persons over the age of 25 years without a high school education
- **Percent Aged Under** 18 or Over 64:Percent of population under 18 or over 64 years of age (`percent_aged_under_18_or_over_64`): (ie. dependents)
- **Per Capita Income** (`per_capita_income_`): Community Area per capita income is estimated as the sum of tract-level aggragate incomes divided by the total population
- **Hardship Index** (`hardship_index`): Score that incorporates each of the six selected socioeconomic indicators

In this Lab, we'll take a look at the variables in the socioeconomic indicators dataset and do some basic analysis with Python.

Connect to the database

Let us first load the SQL extension and establish a connection with the database

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

[3]:
```
# These libraries are pre-installed in SN Labs. If running in another environment please uncomment lines below to install them:
# !pip install --force-reinstall ibm_db==3.1.0 ibm_db_sa==0.3.3
# Ensure we don't load_ext with sqlalchemy>=1.4 (incompadible)
# !pip uninstall sqlalchemy==1.4 -y && pip install sqlalchemy==1.3.24
# !pip install ipython-sql
```

[4]:
```
%load_ext sql
```

```
[6]: # Remember the connection string is of the format:
     # %sql ibm_db_sa://my-username:my-password@hostname:port/BLUDB?security=SSL
     # Enter the connection string for your Db2 on Cloud database instance below
     # i.e. copy after db2:// from the URI string in Service Credentials of your Db2 instance. Remove the double quotes at the end.
     %sql ibm_db_sa://pgh93728:L41DeyB5BM5zsWjx@55fbc997-9266-4331-afd3-888b05e734c0.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31929/BLUDB?security=
```

```
[6]: 'Connected: pgh93728@BLUDB'
```

Store the dataset in a Table¶

In many cases the dataset to be analyzed is available as a .CSV (comma separated values) file, perhaps on the internet. To analyze the data using SQL, it first needs to be stored in the database.

We will first read the dataset source .CSV from the internet into pandas dataframe

Then we need to create a table in our Db2 database to store the dataset. The PERSIST command in SQL "magic" simplifies the process of table creation and writing the data from a pandas dataframe into the table

```
[7]: import pandas
     chicago_socioeconomic_data = pandas.read_csv('https://data.cityofchicago.org/resource/jcxq-k9xf.csv')
     %sql PERSIST chicago_socioeconomic_data
```

```
 * ibm_db_sa://pgh93728:***@55fbc997-9266-4331-afd3-888b05e734c0.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31929/BLUDB
[7]: 'Persisted chicago_socioeconomic_data'
```

You can verify that the table creation was successful by making a basic query like:

Problems

Problem 1

How many rows are in the dataset?

```
[8]: %sql SELECT COUNT(*) FROM chicago_socioeconomic_data;
```

```
 * ibm_db_sa://pgh93728:***@55fbc997-9266-4331-afd3-888b05e734c0.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31929/BLUDB
Done.
[8]:
```

1
78

▼ Click here for the solution

```
%sql SELECT COUNT(*) FROM chicago socioeconomic data;
```

Problem 2

How many community areas in Chicago have a hardship index greater than 50.0?

[9]:
```
%sql SELECT COUNT(*) FROM chicago_socioeconomic_data WHERE hardship_index > 50.0;
```

```
 * ibm_db_sa://pgh93728:***@55fbc997-9266-4331-afd3-888b05e734c0.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31929/BLUDB
Done.
```

[9]:

1
38

▼ Click here for the solution

```
%sql SELECT COUNT(*) FROM chicago_socioeconomic_data WHERE hardship_index > 50.0;

Correct answer: 38
```

Problem 3

What is the maximum value of hardship index in this dataset?

[11]:
```
%sql SELECT MAX(hardship_index) FROM chicago_socioeconomic_data;
```

```
 * ibm_db_sa://pgh93728:***@55fbc997-9266-4331-afd3-888b05e734c0.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31929/BLUDB
Done.
```

[11]:

1
98.0

▼ Click here for the solution

```
%sql SELECT MAX(hardship_index) FROM chicago_socioeconomic_data;

Correct answer: 98.0
```

Problem 4

Which community area which has the highest hardship index?

[12]:
```
%sql SELECT community_area_name FROM chicago_socioeconomic_data where hardship_index=98.0
```

```
 * ibm_db_sa://pgh93728:***@55fbc997-9266-4331-afd3-888b05e734c0.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31929/BLUDB
Done.
```

[12]:

community_area_name
Riverdale

▼ Click here for the solution

```
#We can use the result of the last query to as an input to this query:
%sql SELECT community_area_name FROM chicago_socioeconomic_data where hardship_index=98.0

#or another option:
%sql SELECT community_area_name FROM chicago_socioeconomic_data ORDER BY hardship_index DESC NULLS LAST FETCH FIRST ROW ONLY;

#or you can use a sub-query to determine the max hardship index:
%sql select community_area_name from chicago_socioeconomic_data where hardship_index = ( select max(hardship_index) from
chicago_socioeconomic_data )

Correct answer: 'Riverdale'
```

Problem 5

Which Chicago community areas have per-capita incomes greater than $60,000?

[14]:
```
%sql SELECT community_area_name FROM chicago_socioeconomic_data WHERE per_capita_income_ > 60000;
```

```
 * ibm_db_sa://pgh93728:***@55fbc997-9266-4331-afd3-888b05e734c0.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31929/BLUDB
Done.
```

[14]:

community_area_name
Lake View
Lincoln Park
Near North Side
Loop

Problem 6

Create a scatter plot using the variables `per_capita_income_` and `hardship_index`. Explain the correlation between the two variables.

[ ]:
```
# if the import command gives ModuleNotFoundError: No module named 'seaborn'
# then uncomment the following line i.e. delete the # to install the seaborn package
# !pip install seaborn==0.9.0

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

income_vs_hardship = %sql SELECT per_capita_income_, hardship_index FROM chicago_socioeconomic_data;
plot = sns.jointplot(x='per_capita_income_',y='hardship_index', data=income_vs_hardship.DataFrame())
```

▼ Click here for the solution

```
# if the import command gives ModuleNotFoundError: No module named 'seaborn'
# then uncomment the following line i.e. delete the # to install the seaborn package
# !pip install seaborn==0.9.0

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

income_vs_hardship = %sql SELECT per_capita_income_, hardship_index FROM chicago_socioeconomic_data;
plot = sns.jointplot(x='per_capita_income_',y='hardship_index', data=income_vs_hardship.DataFrame())

Correct answer:You can see that as Per Capita Income rises as the Hardship Index decreases. We see that the points on the scatter plot
are somewhat closer to a straight line in the negative direction, so we have a negative correlation between the two variables.
```

Conclusion

Now that you know how to do basic exploratory data analysis using SQL and python visualization tools, you can further explore this dataset to see how the variable `per_capita_income_` is related to `percent_households_below_poverty` and `percent_aged_16_unemployed`. Try to create interesting visualizations!

Summary

In this lab you learned how to store a real world data set from the internet in a database (Db2 on IBM Cloud), gain insights into data using SQL queries. You also visualized a portion of the data in the database to see what story it tells.

Summary & Highlights

Congratulations! You have completed this lesson. At this point in the course, you know:

- You can access a database from a language like Python by using the appropriate API. Examples include ibm_db API for IBM DB2, psycopg2 for ProstgreSQL, and dblib API for SQL Server.

- DB-API is Python's standard API for accessing relational databases. It allows you to write a single program that works with multiple kinds of relational databases instead of writing a separate program for each one.

- The DB_API connect constructor creates a connection to the database and returns a Connection Object, which is then used by the various connection methods.

- The 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.

- You can use **SQL Magic** commands to execute queries more easily from Jupyter Notebooks. Magic commands have the general format **%sql select * from tablename**. **Cell magics** start with a double %% (percent) sign and apply to the entire cell. **Line magics** start with a single % (percent) sign and apply to a particular line in a cell

Module Introduction & Learning Objectives

Bookmarked

Module Introduction

In this assignment, you will be working with multiple real-world datasets for the city of Chicago. You will be asked questions that will help you understand the data just as you would in the real wold. You will be assessed on the correctness of your SQL queries and results.

Learning Objectives

- Demonstrate skill in retrieving SQL query results and analyzing data
- Demonstrate effective use of formulating SQL queries
- Demonstrate use of invoking SQL queries from Jupyter notebooks using Python

Working with Real World Data Sets

we'll give you a few hints and tips for working with Real World Data-sets.

Many of the real world data sets are made available as .CSV files. These are text files which contain

data values typically separated by commas. In some cases, a different separator such as a semicolon may be used.

For this video, we will use an example of a file called DOGS.CSV. Although this is a fictional data set that contains names of dogs and their breeds, we will use it to illustrate concepts that you will then apply to real datasets.

Sample contents of the DOGS.CSV file are shown here. The first row in the table in many cases contains attribute labels which map to column names in a table.

In DOGS.CSV, the first row contains the name of three attributes. Id is the name of the first attribute and the subsequent rows contain Id values of 1, 2, and 3. The name of the dog is the second attribute.

In this case the dog names Wolfie, Fluffy, and Huggy are the values. The third attribute is called breed, either the dominant breed or pure breed name. It has values of German Shepherd, Pomeranian, and Labrador.

As we've just seen, CSV files can have the first or a header row that contains the names of the attributes. If you're loading the data into the database using the visual load tool in the database console, ensure the header in first row is enabled.

This will map the attribute names in the first row of the CSV file into column names in the database table, and the rest of the rows into the data rows in the table, as shown here. Note that the default column names may not always be database or query friendly, and if that is the case, you may

want to edit them before the table is created. Now, let's talk about querying column names that are lower or mixed case, that is, a combination of upper and lowercase.

Let's assume we loaded the DOGS.CSV file using the default column names from the CSV.

If we try to retrieve the contents of the Id column using the query, select id from DOGS, we'll get an error as shown indicating the id is not valid. This is because the database parser assumes uppercase names by default.

Whereas when we loaded the CSV file into the database it had the Id column name in mixed case i.e

an uppercase I and a lowercase d. In this case, to select data from a column with a mixed case name, we need to specify the column name in its correct case within double quotes as follows.

Select * "Id" from DOGS. Ensure you use double quotes around the column name and not single quotes. Next, we'll cover querying column names that have spaces and other characters.

In a CSV file, if the name of the column contain spaces, by default the database may map them to underscores. For example, in the name of dog column, there are spaces in between the three words.

The database may change it to Name_of_Dog. Other special characters like parentheses or

brackets may also get mapped to underscores. Therefore, when you write a query ensure you use proper case formatting within quotes and substitute special characters to underscores as shown in this example.

Select "Id," "Name_of_Dog," "Breed__dominant_breed_if_not_pure_breed_"from dogs.

Please note the underscores separating the words within double quotes. Also note the double underscore between breed and dominant as shown. Finally, it's also important to note the trailing underscore after the word breed near the end of the query. This is used in place of the closing bracket. When using quotes in Jupyter notebooks, you may be issuing queries in a notebook by first assigning them to Python variables.

In such cases, if your query contains double quotes for example, to specify a mixed case column name, you could differentiate the quotes by using single quotes for the Python variable to enclose this SQL query and double quotes for the column names.

For example, selectQuery ='select "Id" from dogs.' Now, what if you need to specify single quotes within the query, for example, to specify a value in the where clause? In this case you can use backslash as the escape character as follows, select Query = 'select * from dogs where "Name_of_Dog"=\'Huggy\' '. If you have very long queries such as join queries or nested queries, it may be useful to split the query into multiple lines for improved readability.

In Python notebooks, you can use the backslash character to indicate continuation to the next row

as shown in this example. %sql select "Id," Name_of_Dog," \ from dogs \ where"Name_of_Dog" = 'Huggy.' It would be helpful at this point to take a moment to review the special characters as shown.

Please keep in mind that you might get an error if you split the query into multiple lines in

a Python notebook without the backslash. When using SQL magic, you can use the double percent SQL in the first line of the cell in Jupyter Notebooks. It implies that the rest of the content of

the cell is to be interpreted by SQL magic. For example %% sql new row select "Id", "Name_of_dog," new row, from dogs, new row, where "Name_of_dog = 'Huggy.' Again, please note the special characters as shown. When using %% sql the backslash is not needed at the end of each line.

At this point you might be asking, how would you restrict the number of rows retrieved?

It's a good question because a table may contain thousands or even millions of rows, and you may only want to see some sample data or look at just a few rows to see what kind of data the table contains.

You may be tempted to just do select * from table name to retrieve the results in a Pandas data frame and do a head function on it. But, doing so may take a long time for a query to run. Instead, you can restrict the results set by using the limit clause. For example, use the following query to retrieve just the first three rows in a table called census data.

Select * from census_data limit 3. In this video we looked at some considerations and tips for working with real-world datasets.

Working with CSV files

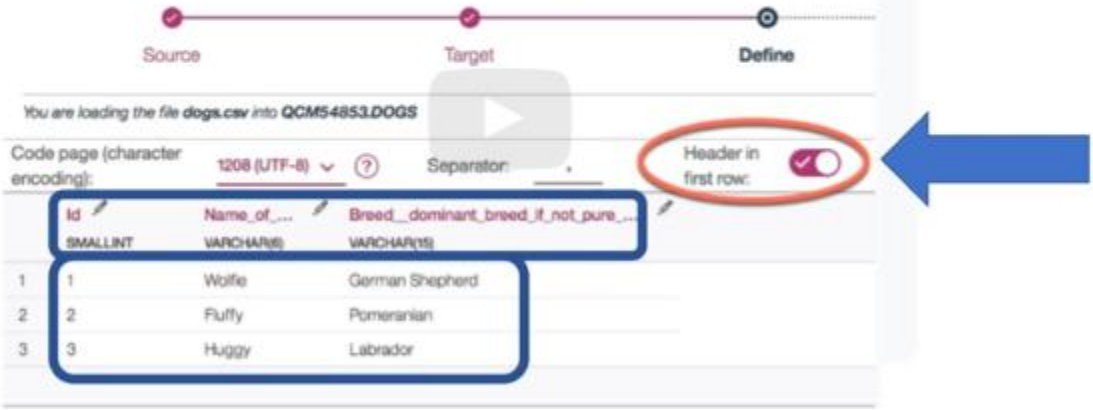
- Many real data sets are .CSV files
- .CSV: COMMA SEPARATED VALUES

- Example: DOGS.CSV

```
Id,Name of Dog,Breed (dominant breed if not pure breed)
1,Wolfie,German Shepherd
2,Fluffy,Pomeranian
3,Huggy,Labrador
```

Column names in first row

When header row in CSV file contains column names:



Querying column names with mixed (upper and lower) case

Retrieve Id column from DOGS table. Try:

```
select id from DOGS
```

If you run this query, you will get this error:

```
Error: "ID" is not valid in the context where it is
used.. SQLCODE=-206, SQLSTATE=42703, DRIVER=4.22.36
```

Querying column names with spaces and special characters

By default, spaces are mapped to underscores:

Other characters may also get mapped to underscores:

```
select "Id", "Name_of_Dog",
"Breed__dominant_breed_if_not_pure_breed_"
from dogs
```

Breed (dominant breed if not pure breed)

Using quotes in Jupyter notebooks

First assign queries to variables:

```
selectQuery = 'select "Id" from dogs'
```

Use a backslash \ as the escape character in cases where the query contains single quotes:

```
selectQuery = 'select * from dogs
      where "Name_of_Dog"=\'Huggy\' '
```

Using quotes in Jupyter notebooks

First assign queries to variables:

```
selectQuery = 'select "Id" from dogs'
```

Use a backslash \ as the escape character in cases where the query contains single quotes:

```
selectQuery = 'select * from dogs
      where "Name_of_Dog"=\'Huggy\' '
```

Splitting queries to multiple lines in Jupyter

Use backslash "\" to split the query into multiple lines:

```
%sql select "Id", "Name_of_Dog", \
    from dogs \
    where "Name_of_Dog"='Huggy'
```

Or use %%sql in the first row of the cell in the notebook:

```
%%sql
select "Id", "Name_of_Dog",
    from dogs
    where "Name_of_Dog"='Huggy'
```

Restricting the # of rows retrieved

To get a sample or look at a small set of rows, limit the result set by using the LIMIT clause:


```
select * from census_data LIMIT 3
```

Getting Table and Column Details

we'll look at how to get information about tables and their columns in a database.

Now how would you get a list of tables in the database? Sometimes your database may contain several tables, and you may not remember the correct name.

For example, you may wonder whether the table is called dog, dogs or four legged mammals.

Database systems typically contain system or catalog tables, from where you can query

the list of tables and get their properties. In DB2 this catalog is called syscat tables.

In SQL Server, it's information schema tables, and in Oracle it's all tables or user tables.

To get a list of tables in a DB2 database, you can run the following query.

Select star from syscat tables. This select statement will return too many tables including system tables, so it's better to filter the result as shown here.

Select tabschema, tabname, create underscore time from syscat tables, where tabschema equals ABC12345.

Please ensure that you replace ABC12345 with your own DB2 username.

When you do a select star from syscat tables, you get all the properties of the tables.

Sometimes we're interested in specific properties such as creation time. Let's say you've created

several tables with similar names. For example, dog one, dog underscore test, dog test one and so on. But, you want to check which of these tables was the last one you created; to do so,

you can issue a query like select tabschema, tabname, create underscore time from syscat tables: Where a tabschema equals QCM54853.

The output will contain the schema name, table name, and creation time for all tables in your schema. Next, let's talk about how to get a list of columns in a table.

If you can't recall the exact name of a column for example, whether it had any lowercase characters or an underscore in its name, in DB2 you can issue a query like the one shown here.

Select star from syscat columns where tab name equals dogs. For your information, in my SQL,

you can simply run the command show columns from dogs, or you may want to know specific properties like the datatype and length of the datatype.

In DB2, you can issue a statement like, select distinct name, coltype, length from sysibm,

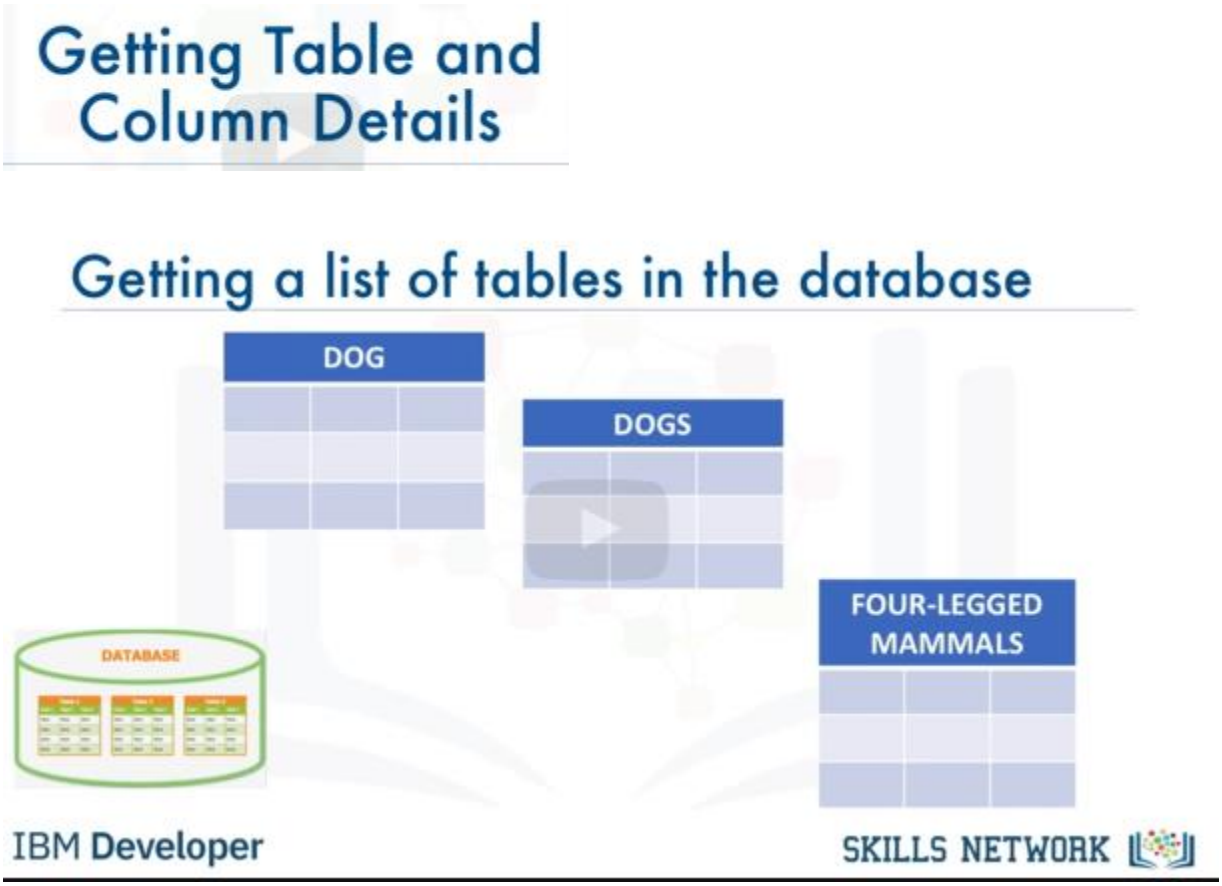
syscolumns where tbname equals dogs. Here we look at the results of retrieving column properties, for a real table called Chicago Crime Data from a Jupyter notebook.

Notice in the output, you can tell certain column names show different cases. For example, the column titled arrest has an uppercase A, and the rest of the characters are lowercase.

So, keep in mind that when you refer to this column in your query, not only must you enclose

the word arrest within double quotes, you must also preserve the correct case inside the quotes.

In this video, we saw how to retrieve table and column information



Getting a list of tables in the database

DB2
SYSCAT.TABLES

SQL Server
INFORMATION_SCHEMA.TABLES

Oracle
ALL_TABLES or USER_TABLES

Getting a list of tables in the database

Query system catalog to get a list of tables & their properties:

```
select * from syscat.tables
```

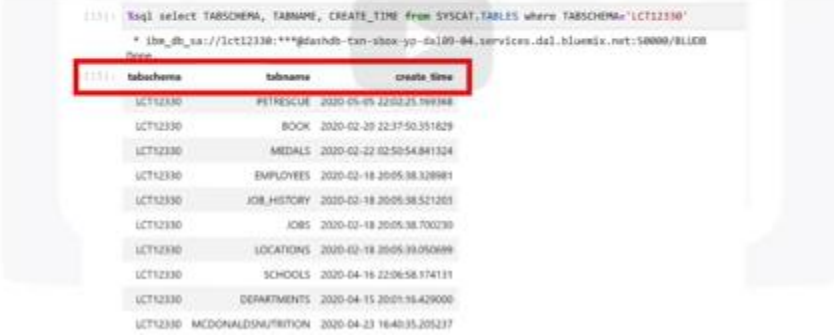

```
select TABSCHEMA, TABNAME, CREATE_TIME
    from syscat.tables
    where tabschema= 'ABC12345'
```

Getting Table Properties

```
select * from syscat.tables
```

Getting Table Properties

```
select TABSCHEMA, TABNAME, CREATE_TIME
     from syscat.tables
     where tabschema='LCT12330'
```



```
%sql select TABSCHEMA, TABNAME, CREATE_TIME from SYSCAT.TABLES where TABSCHEMA='LCT12330'
 * ibm_db_sa://lct12330:***@dashdb-txn-sbox-yp-da09-04.services.dal.bluemix.net:50000/BLUDB
Done.
```

tabschema	tabname	create_time
LCT12330	PETRESCUE	2020-05-05 22:02:25.769348
LCT12330	BOOK	2020-02-20 22:37:50.351629
LCT12330	MEDALS	2020-02-22 02:50:54.841324
LCT12330	EMPLOYEES	2020-02-18 20:05:38.328981
LCT12330	JOB_HISTORY	2020-02-18 20:05:38.521203
LCT12330	JOBS	2020-02-18 20:05:38.700230
LCT12330	LOCATIONS	2020-02-18 20:05:39.050699
LCT12330	SCHOOLS	2020-04-16 22:06:58.174131
LCT12330	DEPARTMENTS	2020-04-15 20:01:16.429000
LCT12330	MCDONALDSNUTRITION	2020-04-23 16:40:35.205237

Getting a list of columns in the database

To obtain the column names query syscat.columns:

```
select * from syscat.columns
       where tabname = 'DOGS'
```

To obtain specific column properties:

```
select distinct(name), coltype, length
       from sysibm.syscolumns
       where tbname = 'DOGS'
```

Column info for a real table

```
In [12]: %sql select distinct(name), coltype, length \
             from sysibm.syscolumns where tbname = 'CHICAGO_CRIME_DATA'

 * ibm_db_sa://qcm54853:***@dashdb-txn-sbox-yp-dal09-04.services.dal
Done.
```

Out[12]:

name	coltype	length
Arrest	VARCHAR	5
Beat	SMALLINT	2
Block	VARCHAR	35
Case_Number	VARCHAR	8
Community_Area	DECIMAL	4
Date	VARCHAR	22
Description	VARCHAR	46
District	DECIMAL	4
Domestic	VARCHAR	5
FBI_Code	VARCHAR	3

Loading Data

Objective(s)

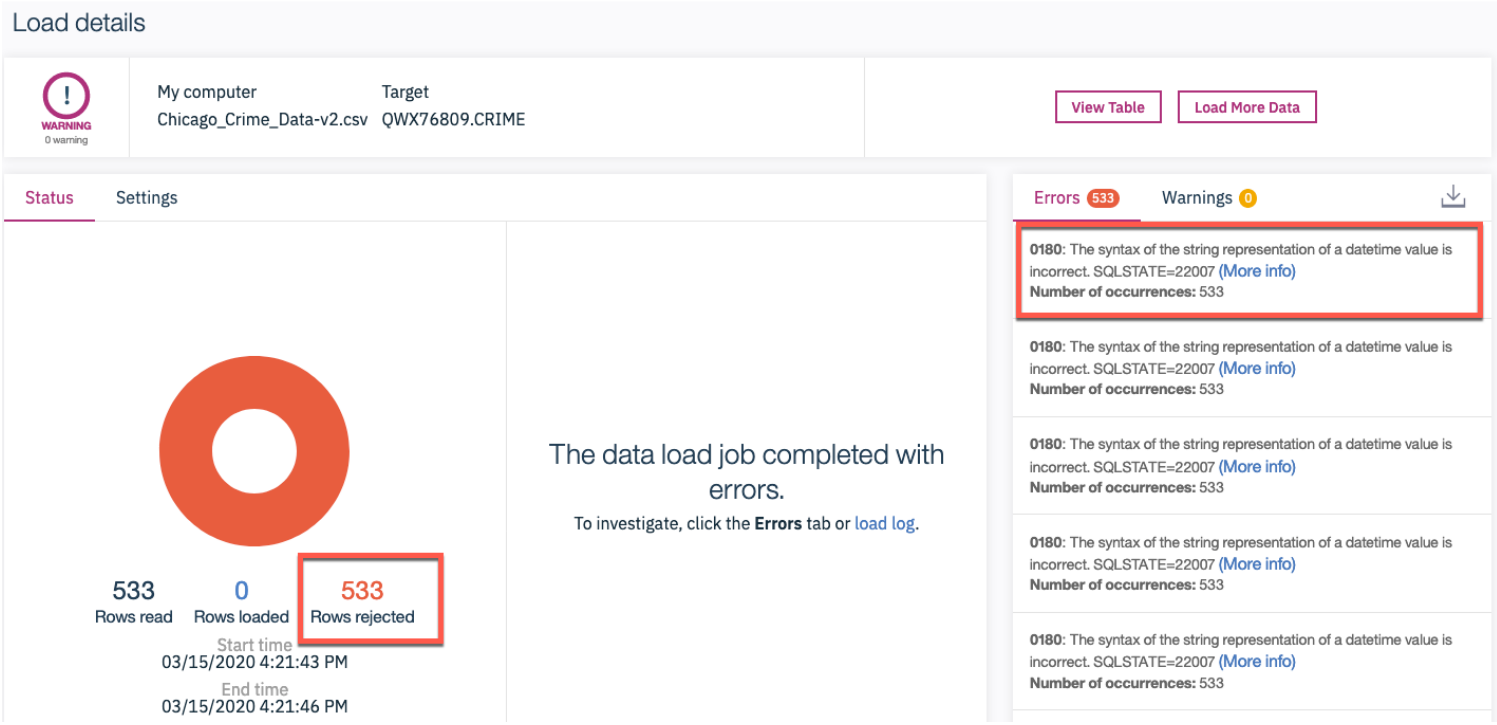
At the end of this lab you will be able to:

- Read data from a CSV files and load it into database

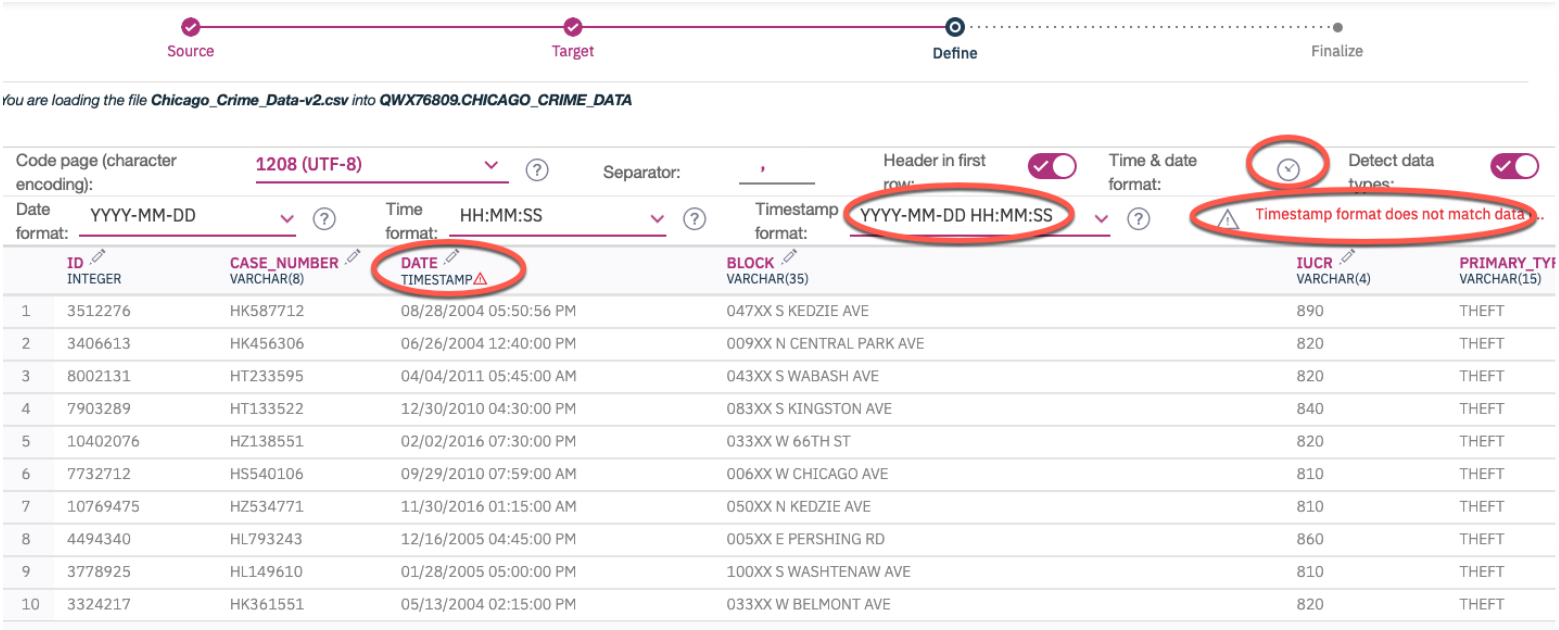
Exercise

When loading data ifrom a CSV file you need to ensure the data in the dataset maps to the correct datatype and format in the database. One area that can be particularly problematic is DATEs, TIMEs, and TIMESTAMPs because their formats can vary significantly.

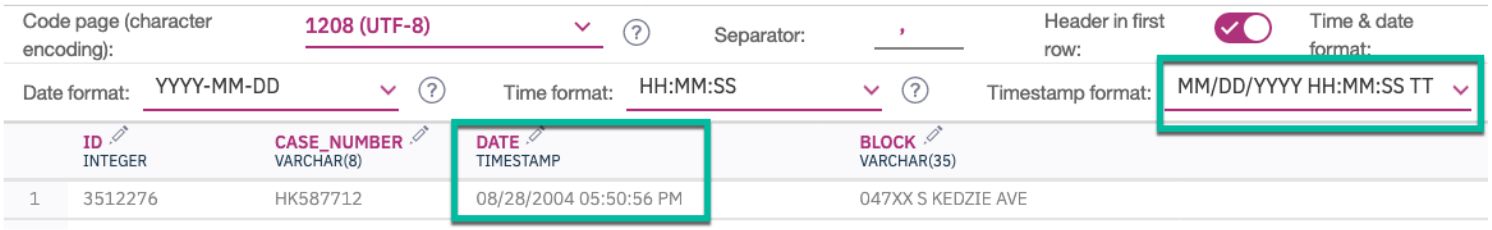
In case the database does not automatically recognize the datatype or format correctly, or the default setting does not match, you will need to manually correct it before loading otherwise you may see an error like the one below when you try to LOAD:



In order to prevent such errors when loading data, in the Db2 console you can preview the datatype and format of the automatically identified values with the values in the datasets in the LOAD screen such as the one below. If there is an issue, it is usually identified with an Warning icon (red triangle with an exclamation mark) next to the datatype of the column (e.g. DATE column in the example below). To correct this, you may first need to click on the "Clock" icon next to the "Time and Date format" to see the formats, if they are not already visible.



First check if there is a pre-defined format in the drop down list that matches the format the date/time/timestamp is in the source dataset. If not, type the correct format. Upon doing so, the Mismatch Warning (and exclamation sign) should disappear. In this example below we changed/overwrote the default Timestamp format of **YYYY-MM-DD HH:MM:SS **to **MM/DD/YYYY HH:MM:SS TT** to match the value of **08/28/2004 05:50:56 PM** in the dataset.



Good luck!