

## Contents

Introduction.....	2
DBMS & RDBMS .....	4
Basic SQL Commands.....	4
COUNT.....	7
DISTINCT .....	8
LIMIT.....	9
INSERT Statement.....	10
UPDATE & DELETE Statements.....	12
Relational Database Concepts.....	18
Relational Model.....	18
Entity-Relationship Model.....	19
Mapping Entity Diagrams to Tables .....	19
Primary Keys and Foreign Keys .....	20
How to create a Database instance on Cloud.....	21
Cloud databases.....	22
<b>Hands-on Lab: Sign up for IBM Cloud, Create Db2 service instance and get started with the Db2 console .....</b>	27
<b>Exercise 1: Sign up for IBM Cloud .....</b>	28
<b>Exercise 2: Create an instance of IBM Db2 Lite plan .....</b>	30
<b>Exercise 3: Locate and Explore the Db2 console .....</b>	31
Types of SQL statements (DDL vs. DML).....	36
CREATE TABLE Statement .....	37
ALTER, DROP, and Truncate tables.....	39
<b>Hands-on Lab: CREATE, ALTER, TRUNCATE, DROP .....</b>	43
<b>Exercise 1: CREATE .....</b>	44
<b>Exercise 2: ALTER.....</b>	46
<b>Exercise 3: TRUNCATE .....</b>	48
<b>Exercise 4: DROP .....</b>	49
<b>Hands-on Lab: Create Tables using SQL Scripts and Load Data into Tables .....</b>	49
<b>Exercise 1: Create tables using SQL scripts .....</b>	49
<b>Exercise 2: Load data into tables .....</b>	52
Using String Patterns and Ranges.....	57
Sorting Result Sets.....	60
Grouping Result Sets .....	62

<b>Hands-on Lab: String Patterns, Sorting and Grouping .....</b>	64
<b>Exercise 1: String Patterns .....</b>	64
<b>Exercise 2: Sorting .....</b>	67
<b>Exercise 3: Grouping .....</b>	69
Built-in Database Functions.....	73
Date and Time Built-in Functions.....	78
<b>Hands-on Lab: Built-in functions - Aggregate, Scalar, String, Date and Time Functions .....</b>	79
<b>Exercise 1: Create the Pet Rescue table.....</b>	79
<b>Exercise 2: Aggregate Functions .....</b>	81
<b>Exercise 3: Scalar and String Functions .....</b>	82
<b>Exercise 4: Date and Time Functions .....</b>	82
Sub-Queries and Nested Selects .....	83
<b>Hands-on Lab: Sub-queries and Nested SELECTs .....</b>	85
<b>Exercise:.....</b>	86
<b>Solution Script.....</b>	90
Working with Multiple Tables .....	90
<b>Hands-on Lab: Working with Multiple Tables .....</b>	96
<b>Exercise 1: Accessing Multiple Tables with Sub-Queries.....</b>	96
<b>Exercise 2: Accessing Multiple Tables with Implicit Joins .....</b>	98
How to Access Databases Using Python .....	101
Writing code using DB-API.....	102
Connecting to a database using ibm_db API .....	106
<b>Create credentials to access your database instance .....</b>	107
<b>Hands-on Lab: Connecting to a database instance .....</b>	110

## Introduction

## Introduction to Databases

### Course Overview

- Basics of SQL
- Relational Database Model
- At the end of this course, you will be able to discuss SQL basics and explain aspects of the relational database model
- At the end of this lesson, you will be able to:
  - Describe SQL, data, database, relational database
  - List five basic SQL commands

## What is SQL?

- A language used for relational databases
- Query data

## What is data?

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



## What is a database?

- A repository of data
- Provides the functionality for adding, modifying and querying that data
- Different kinds of databases store data in different forms

## 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 & RDBMS

### 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

## Basic SQL Commands

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

### SELECT Statement

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

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

### Retrieving rows from a table

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

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

Select \* from <tablename>

# Using the SELECT Statement



Title	Edition	Year	Price	ISBN	Pages	Aisle	Description
Database Fundamentals	1	2010	24.99	978-0-98006283-1	300	DB-A02	Teaches you the fundamentals of databases
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

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	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	978-0-98006283-1	300	DB-A02	Teaches you the fundamentals of databases
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

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

db2 => select book\_id, title, edition, year, price, ISBN, pages, aisle, description from Book

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

4 record(s) selected.

# Retrieving a subset of the columns

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

```

db2 => select book_id, title from Book
Book_ID      Title
B1           Getting started with DB2 Express-C
B2           Database Fundamentals
B3           Getting started with DB2 App Dev
B4           Getting started with WAS CE
  
```

## Restricting the Result Set: WHERE Clause

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

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

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

Book_ID	Title
B1	Getting started with DB2 Express-C

1 record(s) selected

## WHERE Clause Comparison Operators

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

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

## COUNT

## COUNT

COUNT() - a built-in function that retrieves the number of rows matching the query criteria.

Number of rows in a table:

```
select COUNT(*) from tablename
```

## COUNT

Rows in the MEDALS table where Country is Canada:

```
select COUNT(COUNTRY) from MEDALS  
      where COUNTRY='CANADA'
```

Result:

1
-----
29

## DISTINCT

## DISTINCT

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

Retrieve unique values in a column:

```
select DISTINCT columnname from tablename
```

## DISTINCT

List of unique countries that received GOLD medals:

```
select DISTINCT COUNTRY from MEDALS  
      where MEDALTYPE = 'GOLD'
```

Result:

```
1  
---  
21
```

## LIMIT

## LIMIT

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

Retrieve just the first 10 rows in a table:

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

## LIMIT

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

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

Result:

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

## INSERT Statement

## INSERT Statement

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

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

## Adding rows to a table

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



## 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](mailto: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](mailto:rfc@ibm.com)', 'Toronto', 'CA')

('A2', 'Ahuja', 'Rav', '[ra@ibm.com](mailto:ra@ibm.com)', 'Toronto', 'CA')

## UPDATE & DELETE Statements

# UPDATE & DELETE Statements

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

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

## Altering rows of a table – UPDATE statement

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

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

## Using the UPDATE Statement

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

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

## Using the UPDATE Statement

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

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

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

## Using the DELETE Statement

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

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



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

How does the syntax of an INSERT statement look?

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

How does the syntax of an UPDATE statement look?

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

**WHERE** condition;

## How does the syntax of a DELETE statement look?

**DELETE FROM** table\_name

**WHERE** condition

;

1.

1. Problem:

*Update the city for Sandip to Toronto.*

2. Solution:

3. **UPDATE** Instructor

4. **SET** city='Toronto'

5. **WHERE** firstname='Sandip';

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

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

8. **SELECT \* FROM** Instructor;

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

## Practice SQL

Database: Instructors

```
1 SELECT * FROM Instructor;
```

Tip: Autocomplete with Ctrl+Enter or Cmd+Enter

[Submit query](#)

## Results

All commands ran successfully

Support

```
SELECT * FROM Instructor
```

ins_id	lastname	firstname	city	country
1	Ahuja	Rav	Toronto	CA
2	Chong	Raul	Toronto	CA
3	Vasudevan	Hima	Chicago	US
4	Saha	Sandip	Toronto	CA
5	Doe	John	Sydney	AU
6	Doe	Jane	Dhaka	BD
7	Cangiano	Antonio	Vancouver	CA
8	Ryan	Steve	Bariby	GB
9	Sannareddy	Ramesh	Hyderabad	IN

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

1. Problem:

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

2. Solution:

3. **UPDATE** Instructor

4. **SET** city='Dubai', country='AE'

5. **WHERE** ins\_id=5;

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

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

8. **SELECT \* FROM** Instructor;

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

The screenshot shows a 'Practice SQL' interface. At the top, it says 'Database: Instructors'. Below that is a code input box containing '1 SELECT \* FROM Instructor;'. A tip 'Tip: Autocomplete with Ctrl+Enter or Cmd+Enter' is shown above a 'Submit query' button. The results section shows a green bar indicating 'All commands ran successfully'. The results table displays the following data:

ins_id	lastname	firstname	city	country
1	Ahuja	Rav	Toronto	CA
2	Chong	Raul	Toronto	CA
3	Vasudevan	Hima	Chicago	US
4	Saha	Sandip	Toronto	CA
5	Doe	John	Dubai	AE
6	Doe	Jane	Dhaka	BD
7	Cangiano	Antonio	Vancouver	CA
8	Ryan	Steve	Barlby	GB
9	Sannareddy	Ramesh	Hyderabad	IN

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

1. Problem:

*[Remove the instructor record of Doe whose id is 6.]*

2. Solution:

3. **DELETE FROM** instructor
4. **WHERE** ins\_id = **6**;

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

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

7. **SELECT \* FROM** Instructor;

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

The screenshot shows a web-based SQL practice environment. At the top, there's a navigation bar with links to 'home', 'Practice SQL', and 'Instructors'. Below it, a title 'Practice SQL' is displayed. A section titled 'Database: Instructors' shows a code editor with the query 'SELECT \* FROM Instructor;'. A tip at the bottom of the editor says 'Tip: Autocomplete with Ctrl+Enter or Cmd+Enter'. A 'Submit query' button is located below the editor. The results section, titled 'Results', shows a green banner stating 'All commands ran successfully'. The query results are displayed in a table:

ins_id	lastname	firstname	city	country
1	Ahuja	Rav	Markham	CA
2	Chong	Raul	Toronto	CA
3	Vasudevan	Hima	Chicago	US
4	Saha	Sandip	Dhaka	BD
5	Doe	John	Dubai	AE
7	Cangiano	Antonio	Vancouver	CA
8	Ryan	Steve	Barby	GB
9	Sannareddy	Ramesh	Hyderabad	IN

## Relational Database Concepts

### Relational Database Concepts

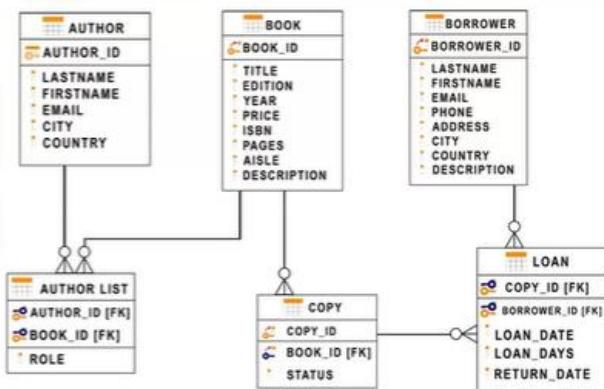
#### Information Model and Data Models

- 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
  - Describe the function of Primary Keys

## Relational Model

# Relational Model

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



logical data independence - physical data independence - physical storage independence

## Entity-Relationship Model

### Entity-Relationship Model

- Used as a tool to design relational databases



BOOK	
	BOOK_ID
■	TITLE
■	EDITION
■	YEAR
■	PRICE
■	ISBN
■	PAGES
■	AISLE
■	DESCRIPTION

## Mapping Entity Diagrams to Tables

# Mapping Entity Diagrams to Tables

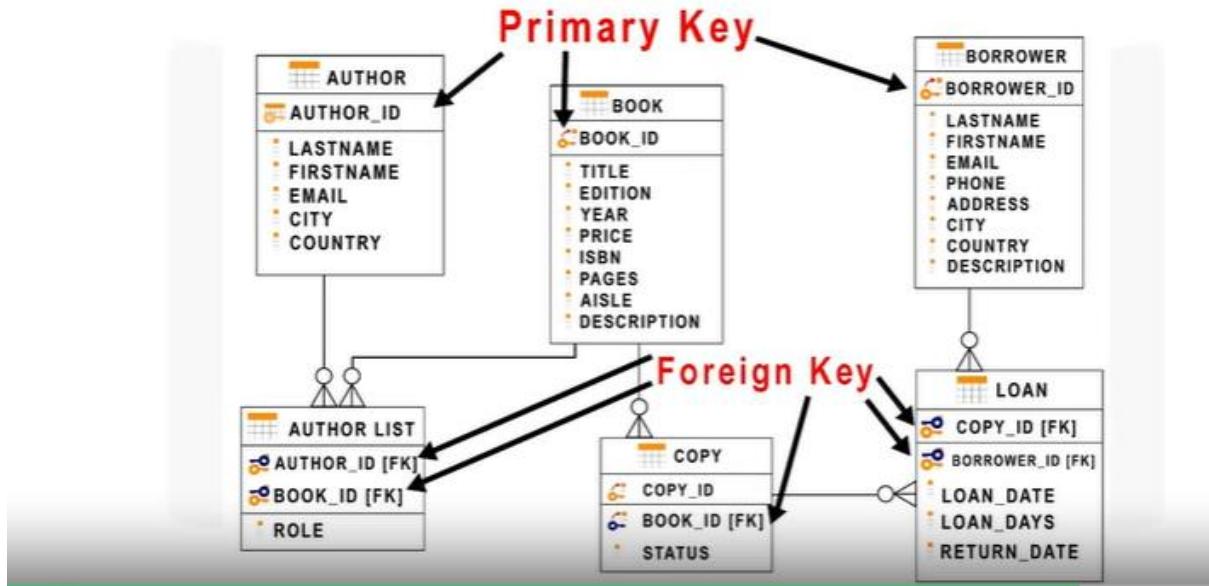
- Entities become tables
- Attributes get translated into columns

Table: Book

Title	Edition	Year	Price	ISBN	Pages	Aisle	Description
Database Fundamentals	1	2010	24.99	978-0-9866283-1-1	300	DB-A02	Teaches you the fundamentals of databases
Getting started with DB2 Express-C	1	2010	24.99	978-0-9866283-5-1	280	DB-A01	Teaches you the essentials of DB2 using DB2 Express-C, the free version of DB2

## Primary Keys and Foreign Keys

## Primary Keys and Foreign Keys



## Summary

Now you know:

- The key advantage of the relational model is data independence
- Entities are independent objects which have Attributes
- Entities map to Tables in a Relational Database
- Attributes map to Columns in a Table
- Common data types include characters, numbers, and dates/times
- A Primary Key uniquely identifies a specific row in a table

## How to craete a Database instance on Cloud

## How to create a Database instance on Cloud

---

### In this video...

- Cloud Database Basics
- List some Cloud Databases
- Describe a Database Instance
- Create an instance of IBM Db2 on Cloud

#### Cloud databases

# Cloud databases

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



## Examples of Cloud databases

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



Available as:

- VMs or Managed Service
- Single or Multi-tenant

# 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



## Deploy an instance of Db2 on Cloud Service

The screenshot shows the IBM Cloud Catalog interface. On the left, a sidebar lists categories: Compute, Containers, Networking, Storage, AI, Analytics, and Databases (which is checked). The main area displays several service options:

- Compose for RethinkDB**: IBM • Analytics • Databases. Description: RethinkDB is a JSON document based, distributed database with an integrated administration and exploration console.
- Compose for ScyllaDB**: IBM • Analytics • Databases. Description: ScyllaDB is a highly performant, in-place replacement for the Cassandra wide-column distributed database.
- Db2**: IBM • Analytics • Databases. Description: A next generation SQL database. Formerly dashDB For Transactions. This option is circled in green and has a red circle around it.
- Db2 Hosted**: IBM • Databases. Description: Db2 Hosted: Offers customers the rich features of an on-premises Db2 deployment without the complexity... (marked with a large red X)
- Db2 Warehouse**: IBM • Analytics • Data. Description: Db2 Warehouse on Cloud offers flexible and powerful data warehousing and enterprise-level analytics. (marked with a large red X)
- GEO Web Services**: Third party • Databases. Description: Adding geo-intelligence to your business. (marked with a blue circle)

1. You can use Db2 on Cloud just as you would use any database software, but without the overhead and expensive hardware setup or software installation and maintenance. Now let's see how we can set up a service instance of Bb2.
2. Navigate to IBM Cloud catalog and select the Db2 service. Note there are several variations of the Db2 service, including Db2 Hosted and Db2 Warehouse. For our purposes, we will choose the Db2 service which comes with a free lite plan.

# Create a new service

Author: IBM • Date of last update: 04/29/2020 • Docs • API docs

Create About

Select a region

Select a region Dallas

Select a pricing plan  
Displayed prices do not include tax. Monthly prices shown are for country or region: [United States](#)

Plan	Features	Pricing
Lite	200 MB of data storage 5 simultaneous connections Shared multitenant system	Free

The Free plan provides a free Db2 service for development and evaluation. The plan has a set amount of limitations as shown. You can continue using the free plan for as long as needed; however, users are asked to re-extend their free account every 90 days by email. If you do not re-extend, your free account is cleaned out a further 90 days later. This helps provide free resources for everyone.

Lite plan services are deleted after 30 days of inactivity.

**Summary**

Db2 Region: Dallas Plan: Lite Service name: Db2-h2

Create Add to estimate

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

# View the newly created service

Dashboard

Resource summary

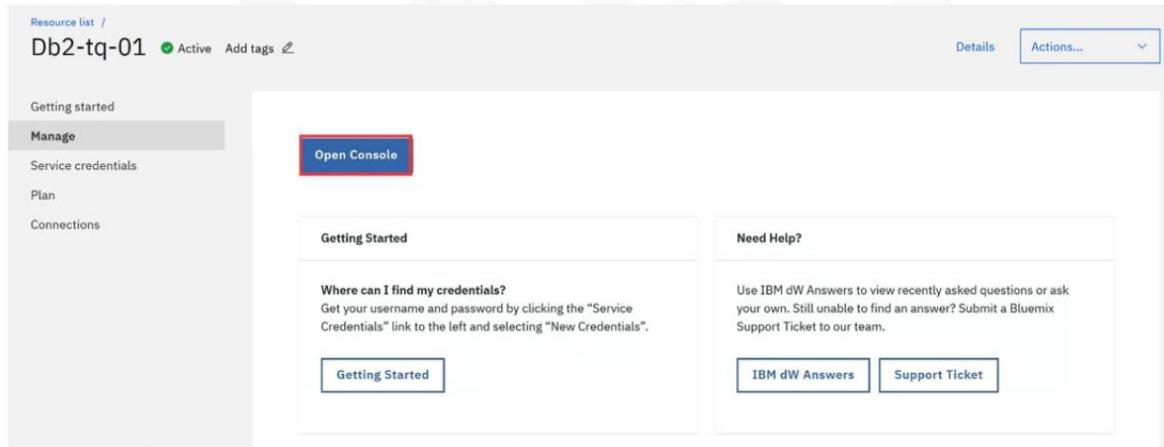
4 Resources

Services (3)

Service	Status
Db2-tq-01	Default
Watson Studio-8w	Default
watson-vision-combined-ey	Default

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

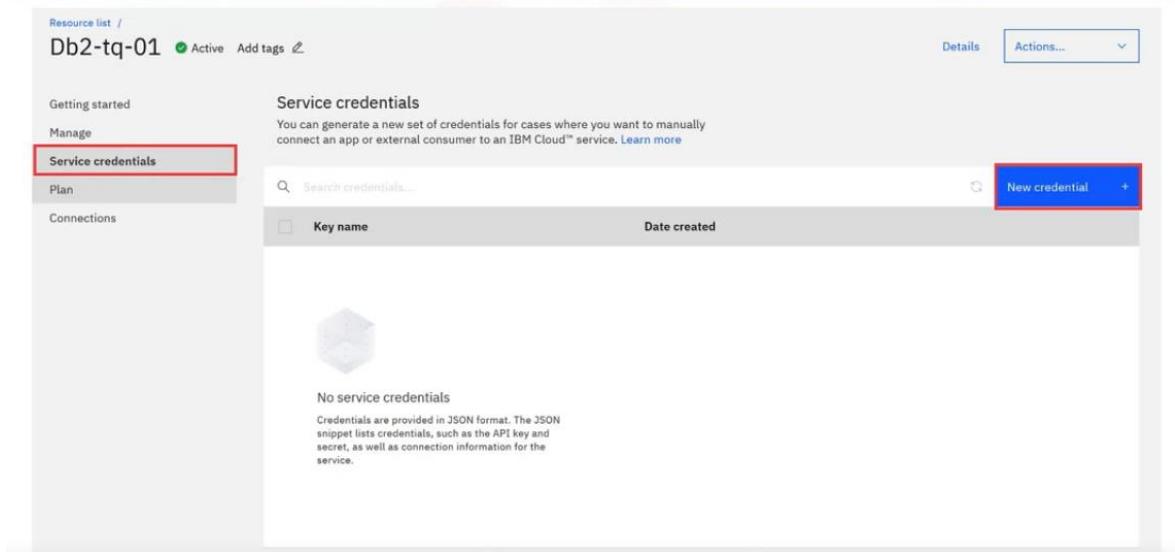
## Manage the database instance



The screenshot shows the IBM Cloud Resource list interface. At the top, it displays 'Resource list / Db2-tq-01 Active Add tags'. Below this, there's a sidebar with tabs: 'Getting started', 'Manage' (which is highlighted), 'Service credentials', 'Plan', and 'Connections'. In the main content area, there's a large blue 'Open Console' button. To its left is a 'Getting Started' section with a 'Getting Started' button. To its right is a 'Need Help?' section with links to 'IBM dW Answers' and 'Support Ticket'. At the bottom right of the main area, there's a 'Details' and 'Actions...' button.

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

## Create new service credentials



The screenshot shows the 'Service credentials' page for the 'Db2-tq-01' service. The sidebar has the 'Service credentials' tab selected (highlighted with a red box). The main content area includes a search bar ('Search credentials...'), a table header ('Key name' and 'Date created'), and a 'New credential' button (also highlighted with a red box). Below the table, there's a note about service credentials and a 'No service credentials' message with a small icon.

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

# 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=zgvr1mlmbzv+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": "*****",
  "port": 50000,
  "ssldsn": "DATABASE=BLUDB;HOSTNAME=dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net;PORT=50001;PROTOCOL=TCPIP;UID=lct12330;PWD=zgvr1mlmbzv+pgg;Security=SSL",
  "ssljdbcurl": "jdbc:db2://dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50001/BLUDB:sslConnection=true",
  "uri": "db2://lct12330:zgvr1mlmbzv%2Bpgg@dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50000/BLUDB",
  "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=zgvr1mlmbzv+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": "*****",
  "port": 50000,
  "ssldsn": "DATABASE=BLUDB;HOSTNAME=dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net;PORT=50001;PROTOCOL=TCPIP;UID=lct12330;PWD=zgvr1mlmbzv+pgg;Security=SSL",
  "ssljdbcurl": "jdbc:db2://dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50001/BLUDB:sslConnection=true",
  "uri": "db2://lct12330:zgvr1mlmbzv%2Bpgg@dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50000/BLUDB",
  "username": "lct12330"
}
```

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

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

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

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

## **Objectives**

After completing this lab, you will be able to:

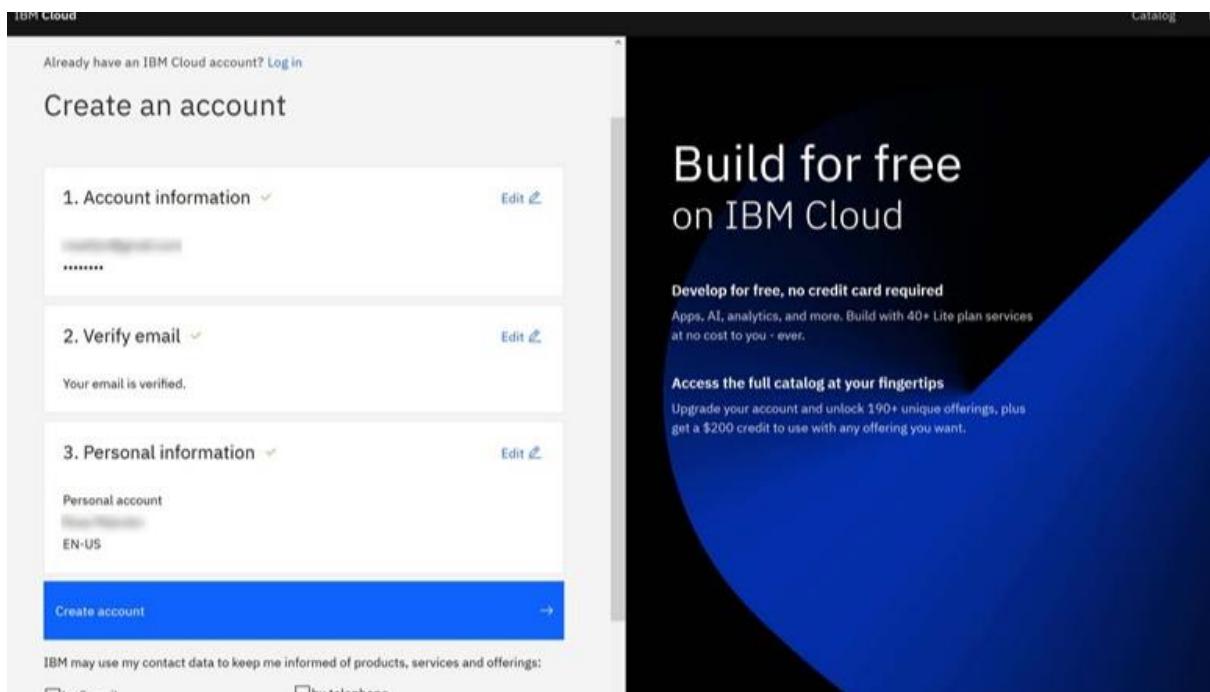
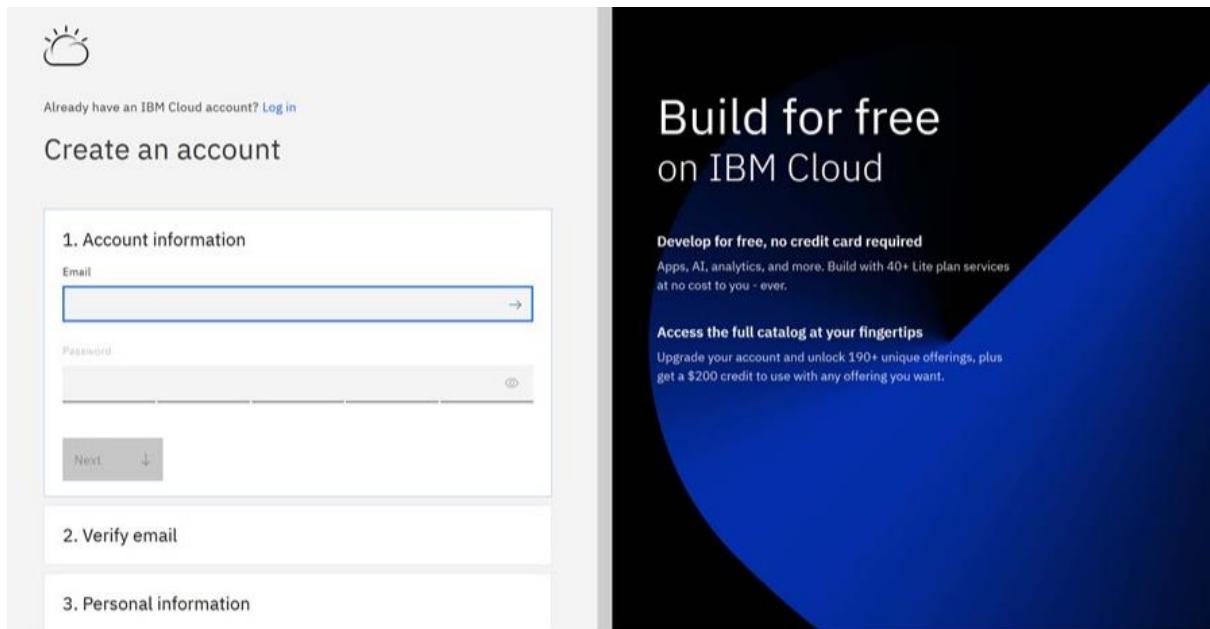
- Sign up for IBM Cloud
- Create an instance of a Db2 service
- Locate and explore the Db2 console

### **Exercise 1: Sign up for IBM Cloud**

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

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

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



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



Hello,

Thank you for signing up for IBM Cloud!

Your 7-digit verification code is:

**1555221**

Enter this verification code on the IBM Cloud registration page where you requested the code. This code is valid for 30 minutes.

Welcome and happy building!

Thank you,  
IBM Cloud

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

### Exercise 2: Create an instance of IBM Db2 Lite plan

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

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

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

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

The screenshot shows the IBM Cloud Catalog interface. A search bar at the top contains the query "cloud.ibm.com/catalog/services/db2?utm\_medium=Exinfluencer&utm\_source=Exinfluencer&utm\_content=000026U&utm\_term=10006555&utm\_id=NA-SkillsNetwork-Channel-SkillsNetwork-CoursesIBMDeveloperSkillsNetworkDB0201ENSkillsNetwork-13214233-2023-02-01". The main page displays a "Db2" service card. The "Create" button is visible at the bottom right of the card. On the right side, there is a summary panel for the "Db2" service, which includes details like Location: London, Plan: Lite, Service name: Db2-xm, and Resource group: Default.

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

This screenshot shows the "Pricing" section of the Db2 service creation page. The "Lite" plan is selected and highlighted with a red box. The "Features" column lists "200 MB of data storage", "5 simultaneous connections", and "Shared multitenant system". The "Pricing" column indicates it is "Free". Below this, a note states: "The Free plan provides a free Db2 service for development and evaluation. The plan has a set amount of limitations as shown. You can continue using the free plan for as long as needed, however, users are asked to re-extract their free account every 90 days by email. If you do not re-extract, your free account is cleaned out a further 90 days later. This helps provide free resources for everyone." A note also says: "Lite plan services are deleted after 30 days of inactivity." To the right, a summary panel shows the service details: Db2, Location: London, Plan: Lite, Service name: Db2-xm, and Resource group: Default. At the bottom right, the "Create" button is highlighted with a red box and is being clicked.

### Exercise 3: Locate and Explore the Db2 console

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

- **NOTE:** You are not required to compose and run any SQL query on this exercise.
1. To access your database instance, go to your IBM Cloud Resource List (you may need to log into IBM Cloud in the process) directly at: [cloud.ibm.com/resources](https://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](https://cloud.ibm.com) and click **Services**.

The screenshot shows the IBM Cloud dashboard. At the top, there's a search bar with placeholder text "Search resources and offerings...". Below it, the word "Dashboard" is displayed. Under "Resource summary", there are two main categories: "Services" (which has a red box around it) and "Storage". The "Services" category has a green checkmark icon with the number "3" next to it, indicating three provisioned services. The "Storage" category has a value of "1". There are also "View resources" and "Create resource" buttons.

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.)

The screenshot shows the IBM Cloud Resource list page. The left sidebar has a tree view with categories like Devices, VPC infrastructure, Clusters, Container Registry, Satellite, Cloud Foundry apps, Cloud Foundry services, and Services and software. The "Services and software" node is expanded, and its first child, "Db2-xm", is highlighted with a red box. The main table lists resources with columns for Name, Group, Location, Product, Status, and Tags. One resource row for "Db2-xm" is visible, showing Default as the Group, London as the Location, Db2 as the Product, and "Provision in progress" as the Status.

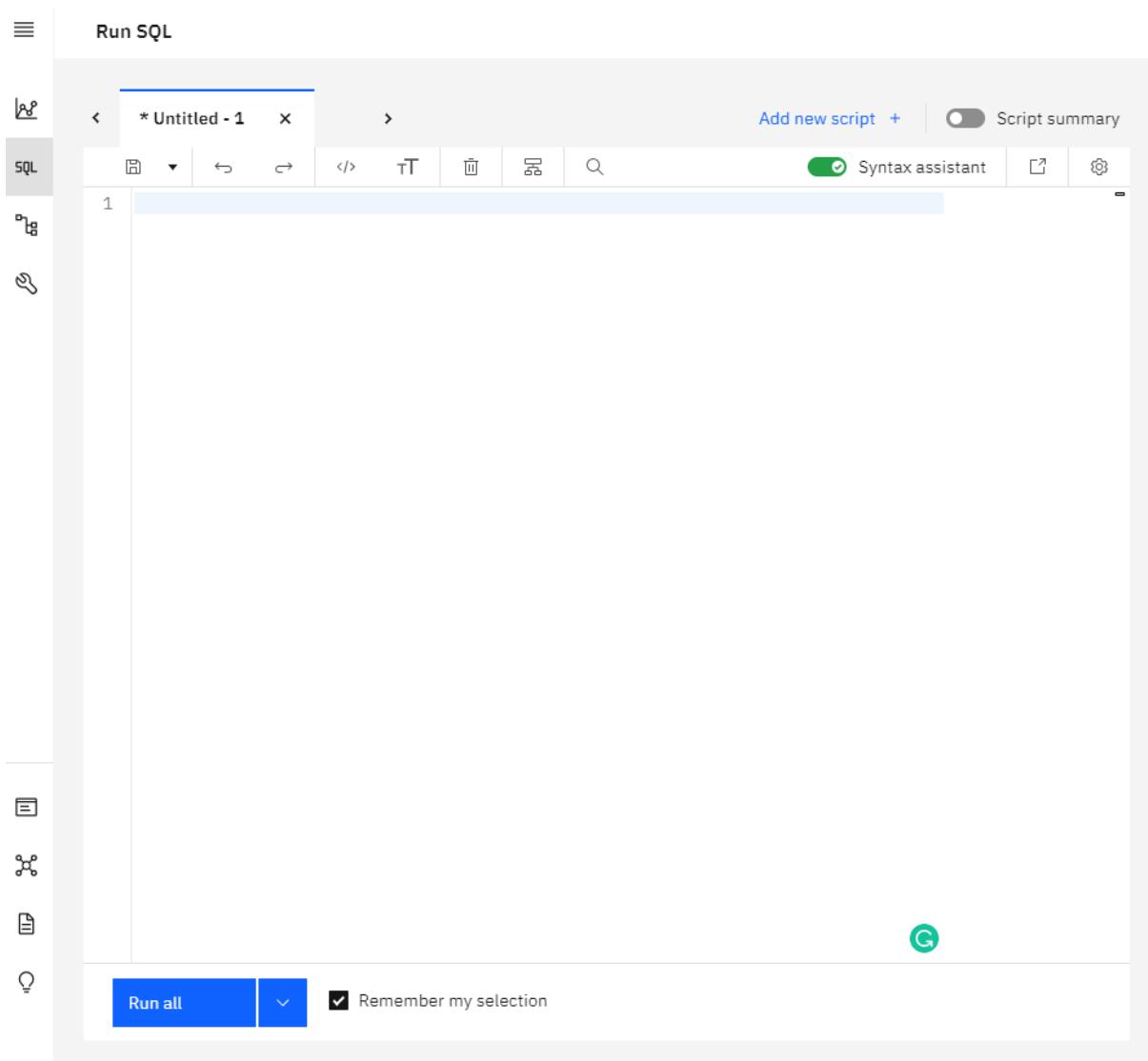
3. Click on the **Go to UI** button.

The screenshot shows the 'Resource list /' section with a resource named 'Db2-xm'. It is marked as 'Active' and has an 'Add tags' button. On the right, there are 'Details' and 'Actions...' buttons. The main content area is titled 'Getting started'. It includes a question 'Where can I find my credentials?' with a note: 'Get your username and password by clicking the "Service Credentials" link to the left and selecting "New Credentials".'. Below this is a red-bordered 'Go to UI' button and a 'Getting started docs' link. A 'Need help?' section follows, with links to 'IBM answers' and 'Support case'.

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**.

The screenshot shows the 'Run SQL' interface. On the left, a sidebar has 'Dashboard', 'SQL' (which is selected and highlighted with a red box), 'Data', and 'Administration' options. The main area is titled 'Run SQL' and contains a 'Choose script so...' dropdown, an 'Open a script to e...' link, a 'From file' button, and a red-bordered 'Create new +' button. Below this is a 'Templates' section with several options: 'Template - SQL Stored ...', 'Template - User Define...', 'Template - Delete Stat...', 'Template - Select Stat...', 'Template - Update Stat...', and 'Template - Insert Stat...'. The 'Create new +' button is also highlighted with a red box.

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



6. The SQL editor will open where you can start typing and running queries.
7. The SQL editor has several areas for performing different tasks.

The screenshot shows a SQL editor interface with the following elements:

- Left Sidebar:** Icons for File, SQL, Table, and Search.
- Top Bar:** Buttons for Undo, Redo, Cut, Copy, Paste, Find, Save, and Run. A "Syntax assistant" toggle is also present.
- Query Editor:** A tab labeled "Untitled - 1" containing the SQL query: "SELECT \* from EMP;". Below it is a button labeled "CLICK HERE TO RUN SQL QUERIES".
- Result Area:** A green checkmark icon with the text "GREEN MARK INDICATES SUCCESS" above the result set. The result set is titled "Result set 1" and contains the following data:

	E1001	JOHN	THOMAS	123456	01/09/1976
1	E1002	Alice	James	123457	07/31/1972
2	E1003	Steve	Wells	123458	08/10/1980
3	E1004	Santosh	Kumar	123459	07/20/1985
4	E1005	Ahmed	Hussain	123410	01/04/1981
5	E1006	Nancy	Allen	123411	02/06/1978

- Bottom Buttons:** "Run all" and "Remember my selection".

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

The screenshot shows the "Add new script" interface with the following elements:

- Top Bar:** Buttons for Undo, Redo, Cut, Copy, Paste, Find, Save, and Run. A "Syntax assistant" toggle is also present.
- Right Side Buttons:** "Add new script +", "Script summary" toggle, and other icons.
- Bottom Buttons:** "From file" and "Create new +". The "Create new +" button is highlighted with a red box.

- Click on **Blank**.

## Add new script

The screenshot shows the "Choose script so..." section with the following elements:

- Top Buttons:** "Choose script so..." and "Open a script to e...".
- Bottom Buttons:** "From file" and "Create new +". The "Create new +" button is highlighted with a red box.

## Templates

Choose a template to start your SQL editor.



- 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.

The screenshot shows the IBM Db2 on Cloud interface. At the top, it says "IBM Db2 on Cloud" and "Storage: 21%". Below that is a "RUN SQL" section with a code editor containing two queries:

```

1 SELECT F_NAME , L_NAME
2 FROM DEPARTMENTS;
3
4 SELECT F_NAME , L_NAME
5 FROM EMPLOYEES;

```

To the right of the code editor is a "Result" pane titled "Result - Dec 11, 2020 4:...". It contains two entries:

- > ✖ SELECT F\_NAME , L\_NAME FROM DEPARTM... Run time: 0.011 s
- > ✓ SELECT F\_NAME , L\_NAME FROM EMPLOYEES Run time: 0.001 s

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.
- o **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.

The screenshot shows the IBM Db2 on Cloud interface. At the top, it says "IBM Db2 on Cloud" and "Storage: 21%". Below that is a "RUN SQL" section with the same two queries as before.

The "Result" pane shows the first query failed:

- > ✖ SELECT F\_NAME , L\_NAME FROM DEPARTM... Run time: 0.011 s

Details for the failed query are shown in a expanded view:

Status: Failed

Error message

"F\_NAME" is not valid in the context where it is used.. SQLCODE=-206, SQLSTATE=42703, DRIVER=4.26.14

[Learn more about this error](#)

The second query succeeded:

- > ✓ SELECT F\_NAME , L\_NAME FROM EMPLOYEES Run time: 0.001 s

The result set for the successful query is displayed in a table:

F_NAME	L_NAME
John	Thomas
Alice	James
Steve	Wells
Santosh	Kumar
Ahmed	Hussain

5 /10 rows truncated to display. [Show More](#)

## Summary

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

## Types of SQL statements (DDL vs. DML)

## Types of SQL Statements - DDL

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

## Summary

Now you know that:

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

## CREATE TABLE Statement

# Objectives

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

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

- Syntax:

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

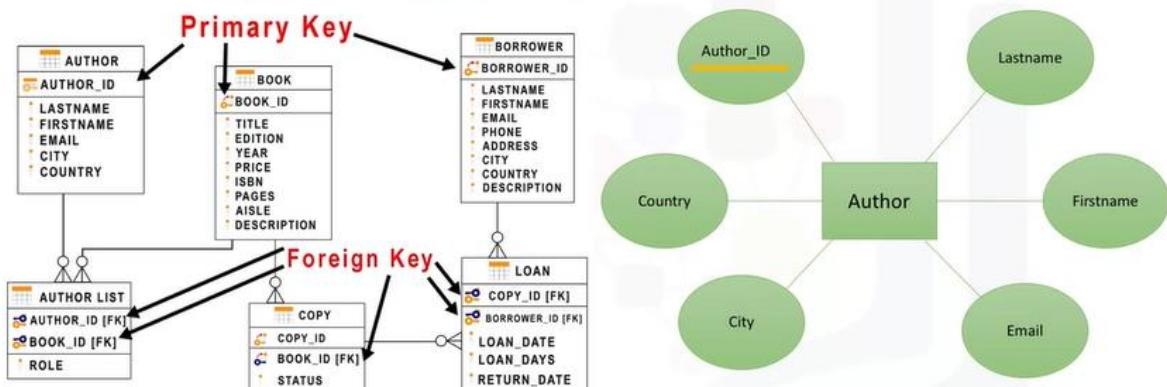
## EXAMPLE

- Create a table for Canadian provinces

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

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

## Create a table



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

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

```
CREATE TABLE author ( author_id CHAR(2) PRIMARY KEY NOT NULL,
lastname VARCHAR(15) NOTNULL,firstname VARCHAR(15) NOTNULL,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.

## 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

# Objectives

After watching this video, you will be able to:

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

## ALTER TABLE ... ADD COLUMN

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

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

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

author_id	lastna me	firstna me	email	city	country	telepho ne_numb er
1001	Thomas	John	johnt@...	New York	USA	5551111
1002	James	Alice	alicej@...	Seattle	USA	5551112
1003	Wells	Steve	stevew:@...	Montreal	Canada	5552222
1004	Kumar	Santosh	kumars@...	London	UK	5553333

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

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

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

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

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

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

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

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

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

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

## DROP TABLE

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

```
DROP TABLE author;
```



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

# TRUNCATE TABLE

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

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

```
TRUNCATE TABLE author
IMMEDIATE;
```

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

author_id	lastname	firstname	email	city	country

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

How does the syntax of a CREATE statement look?

```
CREATE TABLE table_name(
    column1 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;
```

## 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  
)
```

```

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
);

```

- 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;**

ID	PET	SALEPRICE	PROFIT	SALEDATE
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	12.00	2018-06-11
5	Goldfish	48.48	3.5	2018-06-14

ID	ANIMAL	QUANTITY
1	Cat	3
2	Dog	4
3	Hamster	2
1	Cat	3
2	Dog	4

## 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

- 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;
```

```
* Untitled - 2
1 ALTER TABLE PETSALE
2 ADD COLUMN QUANTITY INTEGER;
3
4 SELECT * FROM PETSALE;

Result - Sep 22, 2021 11:02:54 AM
1 ALTER TABLE PETSALE ADD COLUMN QUANTITY INTEGER
Run time: 0.016 s
Status: Success | Affected rows: 0

2 SELECT * FROM PETSALE
Run time: 0.006 s

Result set 1
ID PET SALEPRICE PROFIT SALEDATE QUANTITY
1 Cat 450.09 100.47 2018-05-29
2 Dog 666.66 150.76 2018-06-01
3 Parrot 50.00 8.90 2018-06-04
4 Hamster 60.60 12.00 2018-06-11
5 Goldfish 48.48 3.50 2018-06-14

Result set is truncated, only the first 20 rows have been loaded. Select "View all loaded data" on the right top of the result to view all loaded rows.
More
```

- 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;
```

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

Result - Sep 24, 2021 11:03:44 AM
1 UPDATE PETSALE SET QUANTITY = 9 WHERE ID = 1
Run time: 0.009 s
2 UPDATE PETSALE SET QUANTITY = 3 WHERE ID = 2
Run time: 0.006 s
3 UPDATE PETSALE SET QUANTITY = 2 WHERE ID = 3
Run time: 0.007 s
4 UPDATE PETSALE SET QUANTITY = 6 WHERE ID = 4
Run time: 0.007 s
5 UPDATE PETSALE SET QUANTITY = 24 WHERE ID = 5
Run time: 0.006 s
6
7 SELECT * FROM PETSALE
Run time: 0.006 s

Result set 1
ID PET SALEPRICE PROFIT SALEDATE QUANTITY
1 Cat 450.09 100.47 2018-05-29 9
2 Dog 666.66 150.76 2018-06-01 3
3 Parrot 50.00 8.90 2018-06-04 2
4 Hamster 60.60 12.00 2018-06-11 6
5 Goldfish 48.48 3.50 2018-06-14 24

Result set is truncated, only the first 20 rows have been loaded. Select "View all loaded data" on the right top of the result to view all loaded rows.
More
```

## Task B: ALTER using DROP COLUMN

- 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;
```

```
1 ALTER TABLE PETSALE
2 DROP COLUMN PROFIT;
3
4 SELECT * FROM PETSALE;
```

Result - Sep 22, 2021 11:04:27 AM

**ALTER TABLE PETSALE DROP COLUMN PROFIT**

Status: Success | Affected rows: 0

**SELECT \* FROM PETSALE**

ID	PET	SALEPRICE	SALEDATE	QUANTITY
1	Cat	450.09	2018-05-29	9
2	Dog	666.66	2018-06-01	3
3	Parrot	50.00	2018-06-04	2
4	Hamster	60.60	2018-06-11	6
5	Goldfish	48.48	2018-06-14	24

Result set is truncated, only the first 20 rows have been loaded. Select "View all loaded data" on the right top of the result to view all loaded rows.

## Task C: ALTER using ALTER COLUMN

- 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;
```

```
1 ALTER TABLE PETSALE
2 ALTER COLUMN PET SET DATA TYPE VARCHAR(20);
3
4 SELECT * FROM PETSALE;
```

Result - Sep 22, 2021 11:04:27 AM

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

Status: Success | Affected rows: 0

**SELECT \* FROM PETSALE**

ID	PET	SALEPRICE	SALEDATE	QUANTITY
1	Cat	450.09	2018-05-29	9
2	Dog	666.66	2018-06-01	3
3	Parrot	50.00	2018-06-04	2
4	Hamster	60.60	2018-06-11	6
5	Goldfish	48.48	2018-06-14	24

Result set is truncated, only the first 20 rows have been loaded. Select "View all loaded data" on the right top of the result to view all loaded rows.

- 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.

The screenshot shows a database management interface with three main sections:

- Schemas:** A list of schemas including TPZ00692 (2 tables), AUDIT (0 table), DB2INST1 (0 table), ERRORSCHEMA (0 table), SQL74730 (0 table), and ST\_INFORMTN\_SCHEMA (0 table). "Select All" is checked.
- Tables:** A list of tables under schema TPZ00692: PETRESCUE and PETSALE. PETSALE is selected.
- Table Definition:** Detailed view of the PETSALE table with columns ID, PET, SALEPRICE, SALEDATE, and QUANTITY. The PET column is highlighted with a red border.

## 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;
```

The screenshot shows the execution results of two SQL statements:

- ALTER TABLE PETSALE RENAME COLUMN PET TO ANIMAL;** Status: Success | Affected rows: 0. Run time: 0.019 s.
- SELECT \* FROM PETSALE;** Status: Success | Affected rows: 0. Run time: 0.007 s. Result set 1 shows the following data:

ID	ANIMAL	SALEPRICE	SALEDATE	QUANTITY
1	Cat	450.09	2018-05-29	9
2	Dog	666.66	2018-06-01	3
3	Parrot	50.00	2018-06-04	2
4	Hamster	60.60	2018-06-11	6
5	Goldfish	48.48	2018-06-14	24

Result set is truncated, only the first 20 rows have been loaded. Select "View all loaded data" on the right top of the result to view all loaded rows. More

## 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;
```

The screenshot shows the execution results of two SQL statements:

- TRUNCATE TABLE PET IMMEDIATE;** Status: Success | Affected rows: 0. Run time: 0.015 s.
- SELECT \* FROM PET;** Status: Success | Affected rows: 0. Run time: 0.006 s. Result set 1 shows the following message: You don't have any data currently.

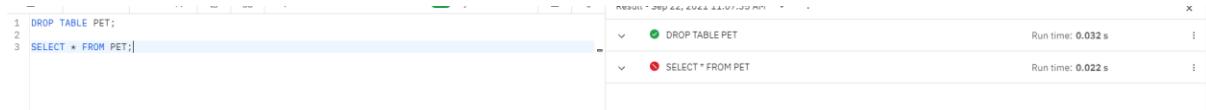
### Exercise 4: DROP

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

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

```
DROP TABLE PET;
```

```
SELECT * FROM PET;
```



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

### Objectives

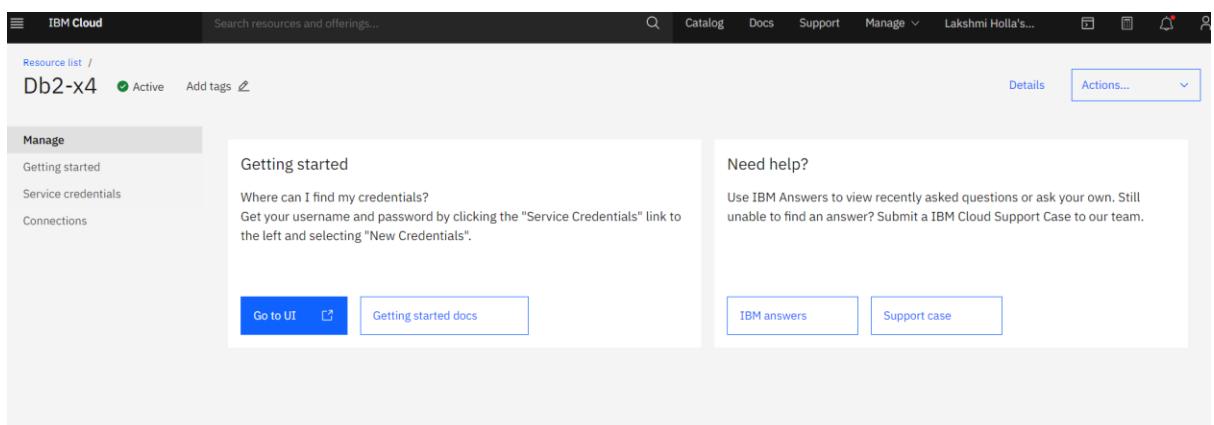
After completing this lab, you will be able to:

- Create tables using SQL scripts
- Load data into tables

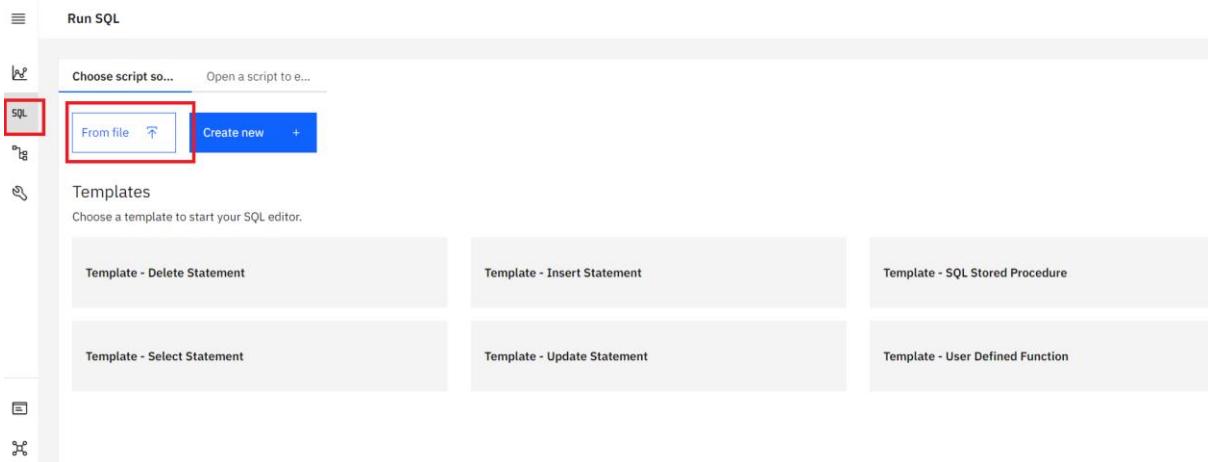
### Exercise 1: Create tables using SQL scripts

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

1. Download the script file to your computer:
  - o [HR Database Create Tables Script.sql](#)
2. Login to IBM Cloud and go to the [Resource List](#) where you can find the Db2 service instance that you created in a previous lab under **Services** section. Click on the Db2-xx service. Next, click on **Go to UI** button.



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



- Locate the file **HR\_Database\_Create\_Tables\_Script.sql** that you downloaded to your computer earlier and open it.
- Once the statements are in the SQL Editor tool , you can run the queries against the database by selecting the **Run All** button.

The screenshot shows the 'Run SQL' interface with a SQL script loaded. The script is titled '\* HR\_Database...'. The code itself is as follows:

```

1  -- DDL statement for table 'HR' database--
2
3  -- Drop the tables in case they exist
4
5  DROP TABLE EMPLOYEES;
6  DROP TABLE JOB_HISTORY;
7  DROP TABLE JOBS;
8  DROP TABLE DEPARTMENTS;
9  DROP TABLE LOCATIONS;
10
11 -- Create the tables
12
13 CREATE TABLE EMPLOYEES (
14     EMP_ID CHAR(9) NOT NULL,
15     F_NAME VARCHAR(15) NOT NULL,
16     L_NAME VARCHAR(15) NOT NULL,
17     SSN CHAR(9),
18     B_DATE DATE,
19     SEX CHAR,
20     ADDRESS VARCHAR(30),
21     JOB_ID CHAR(9),
22     SALARY DECIMAL(10,2),
23     MANAGER_ID CHAR(9),
24     DEP_ID CHAR(9) NOT NULL,
25     PRIMARY KEY (EMP_ID)
26 );
27
28 CREATE TABLE JOB_HISTORY (
29     EMPL_ID CHAR(9) NOT NULL,
30     START_DATE DATE,
31
32

```

At the bottom of the editor, there is a blue 'Run all' button, which is highlighted with a red box. There is also a checked checkbox labeled 'Remember my selection'.

- 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.

```

-- DDL statement for table 'HR' database...
-- Drop the tables in case they exist
DROP TABLE EMPLOYEES;
DROP TABLE JOB_HISTORY;
DROP TABLE JOBS;
DROP TABLE DEPARTMENTS;
DROP TABLE LOCATIONS;

-- Create the tables
CREATE TABLE EMPLOYEES (
    EMP_ID CHAR(9) NOT NULL,
    F_NAME VARCHAR(15) NOT NULL,
    L_NAME VARCHAR(15) NOT NULL,
    SSN CHAR(9),
    B_DATE DATE,
    SEX CHAR,
    ADDRESS VARCHAR(30),
    JOB_ID CHAR(9),
    SALARY DECIMAL(10,2),
    MANAGER_ID CHAR(9),
    DEP_ID CHAR(9) NOT NULL,
    PRIMARY KEY (EMP_ID)
);
CREATE TABLE JOB_HISTORY (
    EMPL_ID CHAR(9) NOT NULL,
    START_DATE DATE,
);

Result - Jul 30, 2021 3:07:47 PM
Status: Failed
Error message
"HYL83142.EMPLOYEES" is an undefined name., SQLCODE=-204, SQLSTATE=42704, DRIVER=4.27.25
Learn more about this error

DML statement for table 'HR' dat...
Run time: 0.035 s

DROP TABLE JOB_HISTORY
Run time: 0.022 s

DROP TABLE JOBS
Run time: 0.024 s

DROP TABLE DEPARTMENTS
Run time: 0.022 s

DROP TABLE LOCATIONS
Run time: 0.025 s

-- Create the tables CREATE TABLE EMPLOYEES (EMP_ID CHAR(9) NOT ...
Run time: 0.214 s

CREATE TABLE JOB_HISTORY (EMPL_ID CHAR(9) NOT NULL, START_DA...
Run time: 0.213 s

CREATE TABLE JOBS (JOB_ID CHAR(9) NOT NULL, JOB_TITLE VAR...
Run time: 0.242 s

CREATE TABLE DEPARTMENTS (DEPT_ID CHAR(9) NOT NULL, DEP...
Run time: 0.261 s

CREATE TABLE LOCATIONS (LOC_ID CHAR(9) NOT NULL, DEP_ID_LOC ...
Run time: 0.290 s

```

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

Name	Type
MYG36304	User

- 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.).

Name	Schema	Properties
DEPARTMENTS	NVG16884	...
EMPLOYEES	NVG16884	...
JOBS	NVG16884	...
JOB_HISTORY	NVG16884	...
LOCATIONS	NVG16884	...
PETSALE	NVG16884	...

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

Name	Data type	Nullable	Length	Scale
DEPT_ID_DEP	CHAR	N	9	0
DEP_NAME	VARCHAR	Y	15	0
MANAGER_ID	CHAR	Y	9	0
LOC_ID	CHAR	Y	9	0

Name	Data type	Nullable	Length	Scale
EMP_ID	CHAR	N	9	0
F_NAME	VARCHAR	N	15	0
L_NAME	VARCHAR	N	15	0
SSN	CHAR	Y	9	0
B_DATE	DATE	Y	4	0
SEX	CHAR	Y	1	0
ADDRESS	VARCHAR	Y	30	0

## Exercise 2: Load data into tables

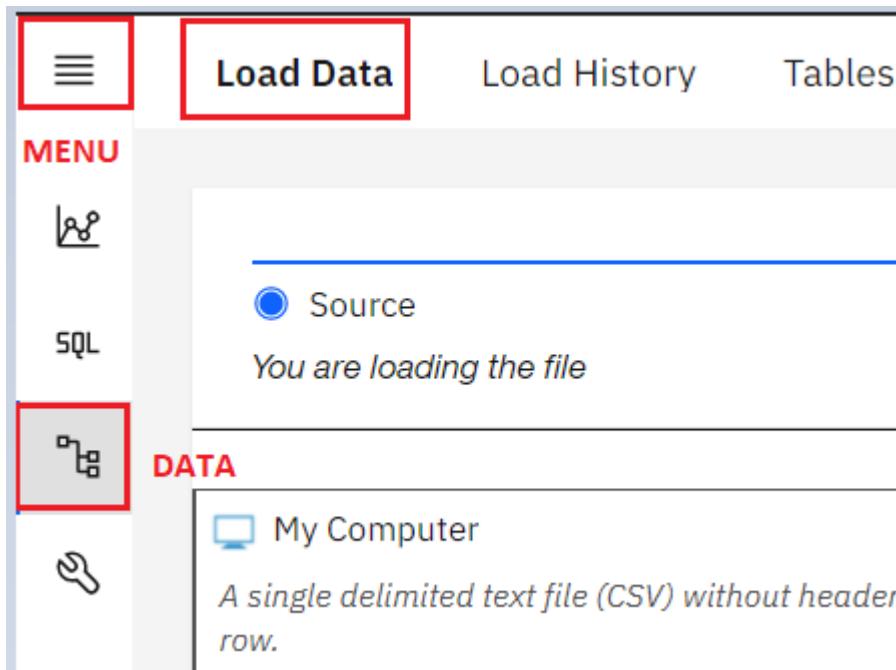
In this exercise, you will learn how data can be loaded into Db2. You could manually insert each row into the table one by one, but that would take a long time. Instead, Db2 (and almost every other database) allows you to load data from .CSV files.

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

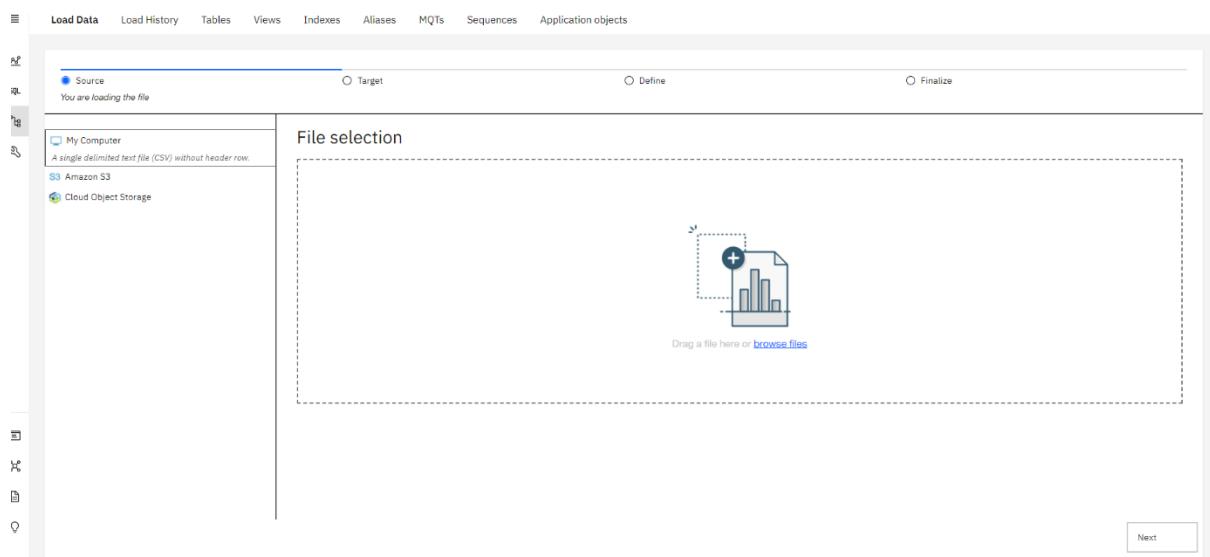
- Download the 5 .csv files below to your local computer:
  - [Departments.csv](#)
  - [Employees.csv](#)

- o [Jobs.csv](#)
- o [Locations.csv](#)
- o [JobsHistory.csv](#)

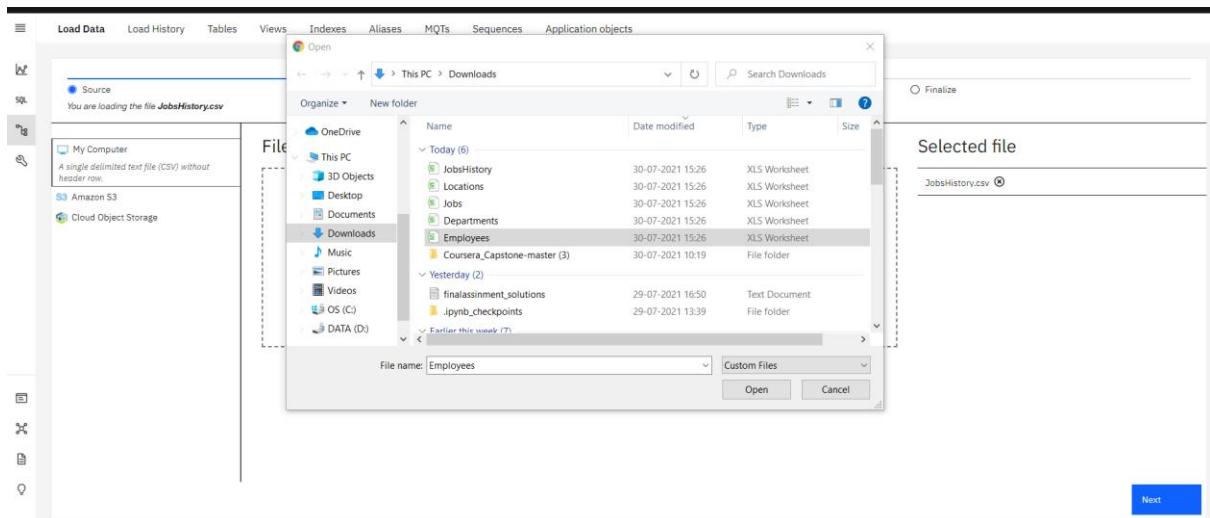
2. In the Db2 Console, from the 3-bar menu icon in the top left corner, click **Load**, and then select **Load Data**.



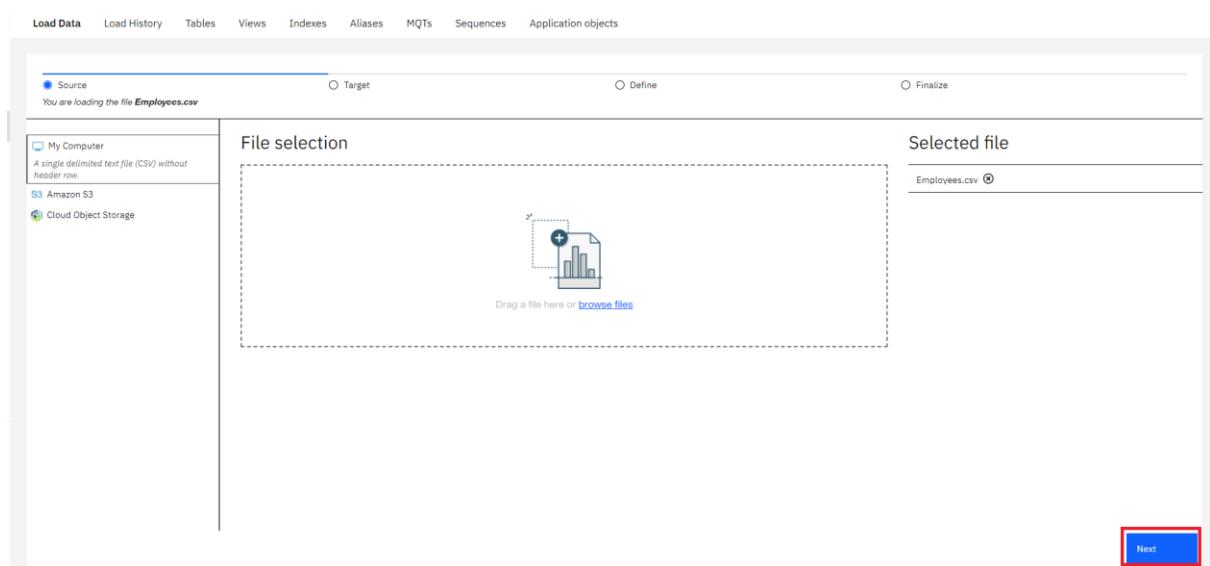
3. On the **Load Data** page that opens, ensure **My Computer** is selected as the source. Click on the **browse files** link.



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



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



- 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.

The screenshot shows the 'Load Data' interface in IBM Db2 on Cloud. The 'Source' tab is selected, and the target is set to 'NVG16884.EMPLOYEES'. In the 'Table' section, 'EMPLOYEES' is selected. On the right, the 'Table definition' pane shows the structure of the EMPLOYEES table. A red box highlights the 'Overwrite table with new data' option under the 'overwrite\_option' section.

- 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.

The screenshot shows the 'Load Data' interface with the 'Target' tab selected. The 'Header in first row' checkbox is unselected. The 'Date format' dropdown is set to 'MM/DD/YYYY'. Other settings like 'Time format' and 'Timestamp format' are also visible. The main area shows a preview of the employee data with columns like EMP\_ID, F\_NAME, L\_NAME, SSN, B\_DATE, SEX, ADDRESS, JOB\_ID, and SALARY.

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

The screenshot shows the 'Review settings' step of a data load process. At the top, it says 'Source' (Employee.csv) and 'Target' (HYL83142.EMPLOYEES). Below this, the 'Summary' section contains various configuration details:

- Code page: 1208 (Default)
- Separator: , (Default)
- Time format: HH:MM:SS (Default)
- Date format: YYYY-MM-DD (Default)
- Timestamp format: YYYY-MM-DD HH:MM:SS (Default)
- String delimiter: (Default)

The 'Option' section shows 'Maximum number of warnings' set to 1000.

At the bottom right are 'Back' and 'Begin Load' buttons.

- 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.

The screenshot shows the 'Load details' step of the data load process. It displays the following information:

- Status:** COMPLETE
- Load details:** My computer → Target: NVG16884.EMPLOYEES
- Statistics:** Rows read: 10, Rows loaded: 10, Rows rejected: 0
- Timing:** Start time: 09/22/2021 11:53:37 AM, End time: 09/22/2021 11:53:40 AM
- Message:** The data load job succeeded. You can now work with your data.
- Notifications:** A green notification states: The data load job succeeded. Load Employees.csv from My Computer to NVG16884.EMPLOYEES. It also shows 'View details' and 'Clear all' buttons.
- Errors and Warnings:** Both are listed as 0.

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

The screenshot shows a database management interface with three main panels:

- Schemas:** Shows a list of schemas, with "NVG16884" selected and highlighted by a red box.
- Tables:** Shows a list of tables in the selected schema. "EMPLOYEES" is selected and highlighted by a red box.
- Table definition:** Shows the detailed structure of the "EMPLOYEES" table, including columns like EMP\_ID, F\_NAME, L\_NAME, SSN, B\_DATE, SEX, ADDRESS, JOB\_ID, SALARY, MANAGER\_ID, and DEP\_ID. A "View data" button is highlighted by a red box.

11. Now you can view the table data.

The screenshot shows the "EMPLOYEES" table data in a grid format:

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

12. Now it's your turn to load data to the remaining 4 tables of the HR database – **LOCATIONS**, **JOB\_HISTORY**, **JOBS**, and **DEPARTMENTS** from the remaining source files.

13. Click **Load More Data** and then follow the steps from **Step 3** above again to load the remaining 4 tables.

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

## Using String Patterns and Ranges

- At the end of this lesson, you will be able to describe how to simplify a SELECT statement by using:

- String patterns
- Ranges, or
- Sets of values

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

## 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	978-0-98666283-5-1	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 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.

`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.

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?

## 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%**

- 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.

## Retrieving rows - using a String Pattern

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

`Firstname`

Raul

Rav

2 record(s) selected.

- 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 are returned for authors Raul and Rav.

## 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

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

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

```
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.

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.

## 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

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

## ORDER BY clause – Descending order

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

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

Ascending order by default

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

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

Descending order with DESC keyword

## Specifying Column Sequence Number

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

Title	Pages
Getting started with WAS CE	278
Getting started with DB2 Express-C	280
Getting started with App Dev	298
Database Fundamentals	300

4 record(s) selected.

Ascending order by Column 2 (number of pages)

## Grouping Result Sets

# Grouping Result Sets

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

## 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.

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.

## 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.

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

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

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

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

6 record(s) selected.

## Restricting the Result Set - HAVING clause

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

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

6 record(s) selected.

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

Country	Count
CN	6
IN	6

6 record(s) selected.

## Summary

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

### Hands-on Lab: String Patterns, Sorting and Grouping

#### Exercise 1: String Patterns

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

1. Problem:

*Retrieve all employees whose address is in Elgin,IL.*

### Hint

Use the LIKE operator to find similar strings.

### Solution

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

### Output

The screenshot shows a database query interface. At the top, there is a code editor window containing the following SQL query:

```
-- Query 1-----  
;  
3 select F_NAME , L_NAME  
4 from EMPLOYEES  
5 where ADDRESS LIKE '%Elgin,IL%' ;  
--Query 2--  
;
```

Below the code editor is a results pane titled "Result". It has tabs for "Saved scripts" and "Result", with "Result" being active. Under "Result", there are tabs for "Filter by status:" (set to "Result set") and "Log". A dropdown menu is open under "Filter by status:" with "All" selected. To the left of the results table is a sidebar with a tree view showing a single node labeled "Al...". Below the tree view are three items: "select...", "select ...", and "select ...". The results table has two columns: "F\_NAME" and "L\_NAME". The data rows are:

F_NAME	L_NAME
Alice	James
Nancy	Allen
Ann	Jacob

At the bottom of the results pane, it says "Total rows: 3".

## 2. Problem:

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

### Hint

Use the LIKE operator to find similar strings.

### Solution

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

### Output

```

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		
<b>Delete All</b>		
<b>▼ All... 🗑</b>		
<b>select ...</b>		
<b>select...</b>		
<b>select ...</b>		
<b>select ...</b>		
Total rows: 4		

### 3. Problem:

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

#### Hint

Use the keyword BETWEEN for this SQL problem.

#### Solution

```

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

```

#### Output

```

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		
<b>Delete All</b>		
<b>▼ All(5)... 🗑</b>		
<b>select F....</b>		
<b>select F....</b>		
<b>select * f...</b>		
Total rows: 2		

## Exercise 2: Sorting

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

### 1. Problem:

*Retrieve a list of employees ordered by department ID.*

Hint

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

Solution

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

Output

red scripts		Result
Query status:		Result set
<hr/>		
All		
<a href="#">Delete All</a>		
<a href="#">(1)...</a>		
<a href="#">select F...</a>		
<a href="#">(1)...</a>		
<a href="#">select F...</a>		
<a href="#">(1)...</a>		
<a href="#">select F...</a>		
<a href="#">(1)...</a>		
<a href="#">select F...</a>		
<a href="#">(1)...</a>		
<a href="#">select F...</a>		
Total rows: 10		

## 2. Problem:

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

### Solution

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

### Output

The screenshot shows a database interface with a sidebar for 'Saved scripts' containing several recent queries, and a main area for the 'Result' of the current query. The 'Result set' tab is selected. The results are displayed in a table with columns: F\_NAME, L\_NAME, and DEP\_ID. The data is as follows:

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

Total rows: 10

## 3. (Optional) Problem:

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

### Solution

```

SELECT D.DEP_NAME , E.F_NAME, E.L_NAME
FROM EMPLOYEES as E, DEPARTMENTS as D
WHERE E.DEP_ID = D.DEPT_ID_DEP
ORDER BY D.DEP_NAME, E.L_NAME DESC;

```

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

## Output

The screenshot shows a database interface with a code editor and a results viewer.

**Code Editor:**

```

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--

```

**Results View:**

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

## Exercise 3: Grouping

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

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

### 1. Problem:

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

## Solution

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

## Output

The screenshot shows a database interface with a query editor and a results viewer. The query in the editor is:

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

The results are displayed in a table titled "Result set". The table has two columns: "DEP\_ID" and a unnamed column. The data is as follows:

DEP_ID	
2	3
5	4
7	3

Total rows: 3

## 2. Problem:

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

## Solution

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

## Output

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

### 3. Problem:

*Label the computed columns in the result set of SQL problem 2 (Exercise 3 Problem 2) as NUM\_EMPLOYEES and AVG\_SALARY.*

## Solution

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

## Output

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

#### 4 Problem:

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

## Solution

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

## Output

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

red scripts	Result
er by status:	Result set
All	<div style="display: flex; align-items: center;"><span style="margin-right: 10px;">▼</span><span style="flex-grow: 1;"></span><span style="margin-left: 10px;">🔍</span><span style="margin-left: 10px;">🖨️</span></div>
ete All	
... 🗑️	
select...	
... 🗑️	
select ...	Total rows: 3

## 5. Problem:

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

### Solution

```
SELECT DEP_ID, COUNT(*) AS "NUM_EMPLOYEES", AVG(SALARY) AS "AVG_SALARY"  
FROM EMPLOYEES  
GROUP BY DEP_ID  
HAVING count(*) < 4  
ORDER BY AVG_SALARY;
```

## Output

```

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;

```

d scripts      Result

by status:      Result set      Log

Delete All

!& F...     

select DEP\_...

select DEP\_I...

!& F...     

Total rows: 2

DEP_ID	NUM_EMPLOYEES	AVG_SALARY
7	3	66666.6666666
2	3	86666.6666666

## Solution Script

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

- [StringPattern-Sorting-Grouping\\_Solution\\_Script.sql](#)

## Built-in Database Functions

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

# PETRESCUE TABLE

## PETRESCUE

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

## Aggregate or Column Functions

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

## SUM

SUM function: Add up all the values in a column

`SUM(COLUMN_NAME)`

Example 1: Add all values in the COST column:

`select SUM(COST) from PETRESCUE`

Example 1: Result:

1

1718.24

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

## Column Alias

Example 2: Explicitly name the output column SUM\_OF\_COST:

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

Example 2: Results:

```
SUM_OF_COST  
1718.24
```

## MIN, MAX

MIN: Return the MINIMUM value

MAX: Return the MAXIMUM value

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

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

Example 3B. Results:

```
1  
24
```

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

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

Example 3B. Results:

```
1  
2
```

# Average

AVG() return the average value

Example 4. Specify the Average value of COST:

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

#### Example 4. Results:

1

## 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

127.270000000000000000000000000000

# SCALAR and STRING FUNCTIONS

**SCALAR:** Perform operations on every input value

Examples: ROUND(), LENGTH(), UCASE, LCASE

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

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

#### Example 6. Results:

**1**  
450.00  
667.00  
100.00  
50.00  
76.00

## UCASE, LCASE

Example 8: Retrieve ANIMAL values in UPPERCASE:

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

Example 8: Results:

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

## UCASE, LCASE

Example 9: Use the function in a WHERE clause :

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

Example 9: Results:

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

## UCASE, LCASE

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

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

Example 10: Results:

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

## Date and Time Built-in Functions

### Date, Time Functions

Most databases contain special datatypes for dates and times.

DATE: YYYYMMDD

TIME: HHMMSS

TIMESTAMP: YYYYXXDDHHMMSSZZZZZ

Date / Time functions:

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

Example 11: Extract the DAY portion from a date:

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

Example 11: Results:

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

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:

	ID	ANIMAL	QUANTITY	COST	RESCUEDATE	+ 3 DAYS
2018-06-01	1	Cat	9	450.09	5/29/2018	6/1/2018
2018-06-04	2	Dog	3	666.66	6/1/2018	6/4/2018
2018-06-07	3	Dog	1	100	6/4/2018	6/7/2018
2018-06-07	4	Parrot	2	50	6/4/2018	6/7/2018
2018-06-13	5	Dog	1	75.75	6/10/2018	6/13/2018

Special Registers:

`CURRENT_DATE, CURRENT_TIME`

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

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

Example 14: Sample result (format YMMDD):

10921

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

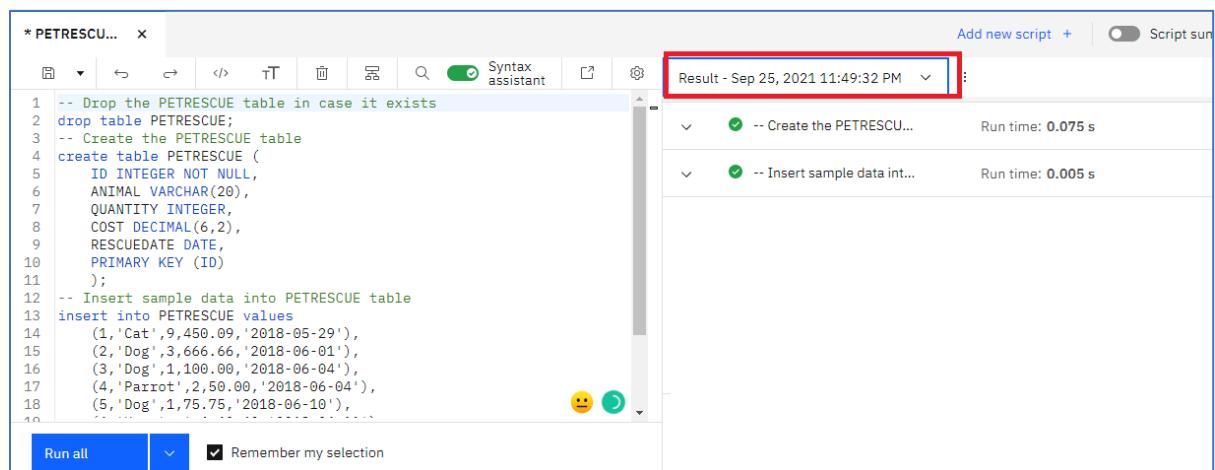
### Exercise 1: Create the Pet Rescue table

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

1. Download the script file [PETRESCUE-CREATE.sql](#)

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

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



The screenshot shows the IBM Cloud DB2 SQL Editor interface. On the left, the script content is displayed:

```

1 -- Drop the PETRESCUE table in case it exists
2 drop table PETRESCUE;
3 -- Create the PETRESCUE table
4 create table PETRESCUE (
5     ID INTEGER NOT NULL,
6     ANIMAL VARCHAR(20),
7     QUANTITY INTEGER,
8     COST DECIMAL(6,2),
9     RESCUEDATE DATE,
10    PRIMARY KEY (ID)
11 );
12 -- Insert sample data into PETRESCUE table
13 insert into PETRESCUE values
14     (1,'Cat',9,450.09,'2018-05-29'),
15     (2,'Dog',3,666.66,'2018-06-01'),
16     (3,'Dog',1,100.00,'2018-06-04'),
17     (4,'Parrot',2,50.00,'2018-06-04'),
18     (5,'Dog',1,75.75,'2018-06-10'),
19

```

On the right, the **Result** section shows the execution details:

- Create the PETRESCUE... Run time: 0.075 s
- Insert sample data int... Run time: 0.005 s

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**.

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

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 datatypes, and so on). You can also click **View Data** to view the content of the table.

NVG16884.PETRESCUE				
ID	ANIMAL	QUANTITY	COST	RESCUEDATE
1	Cat	9	450.09	2018-05-29
2	Dog	3	666.66	2018-06-01
3	Dog	1	100.00	2018-06-04
4	Parrot	2	50.00	2018-06-04
5	Dog	1	75.75	2018-06-10
6	Hamster	6	60.60	2018-06-11
7	Cat	1	44.44	2018-06-11
8	Goldfish	24	48.48	2018-06-14
9	Dog	2	222.22	2018-06-15

## Exercise 2: Aggregate Functions

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

select SUM(COST) from PETRESCUE;

**Query A2:** Enter a function that displays the total cost of all animal rescues in the PETRESCUE table in a column called SUM\_OF\_COST.

select SUM(COST) AS SUM\_OF\_COST from PETRESCUE;

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

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

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

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

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

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

### Exercise 3: Scalar and String Functions

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

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

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

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

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

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

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

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

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

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

### Exercise 4: Date and Time Functions

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

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

**Query C2:** Enter a function that displays the number of rescues on the 5<sup>th</sup> month.

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

**Query C3:** Enter a function that displays the number of rescues on the 14<sup>th</sup> 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

# Sub-queries and Nested Selects

**Sub-query:** A query inside another query

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

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

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

## 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
```

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

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

## Sub-queries to evaluate Aggregate functions

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

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

### Result:

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

## Sub-queries in list of columns

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

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

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

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

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

## 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 ;
```

### 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

### Hands-on Lab: Sub-queries and Nested SELECTs

How does a typical Nested SELECT statement syntax look?

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

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

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

- [Hands-on Lab : Create tables using SQL scripts and Load data into tables](#)

**Exercise:**

1. Problem:

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

Hint

Use the AVG aggregate function.

Solution

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

Output

↙  --- 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);
```

## Output

✓ --- Query 2--- select EMP\_ID, F\_NAME, L\_NAME... Run time: 0.001 s

Result set 1 Search Up Right

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

✓  --- Query 3 --- select EMP\_ID, SALARY, MAX(SA... Run time: 0.005 s

Status: Failed

Error message

An expression starting with "SALARY" specified in a SELECT clause, HAVING clause, or ORDER BY clause is not specified in the GROUP BY clause or it is in a SELECT clause, HAVING clause, or ORDER BY clause with a column function and no GROUP BY clause is specified.. SQLCODE=-119, SQLSTATE=42803, DRIVER=4.26.14

[Learn more about this error](#)

#### 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 Up Right

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

[Show Less](#)

## 5. Problem:

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

### Hint

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

### Solution

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

### Output

✓ --- Query 5 --- select \* from ( select EMP\_ID, F\_NAME, L\_NAME, DEP\_ID from EMPLOYEE )

Run time: 0.001 s

Result set 1

Search



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

Show Less



## Solution Script

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

- [SubQueries\\_Solution\\_Script.sql](#)

## Working with Multiple Tables

# Working with Multiple Tables

Ways to access multiple tables in the same query:

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

## Tables for Examples in this Lesson

EMPLOYEES:

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

DEPARTMENTS:

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

## Accessing Multiple Tables with Sub-queries

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

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

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

**Result:**

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

## Multiple Tables with Sub-queries

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

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

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

**Result:**

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

To retrieve the department ID and name for employees who earn more than \$70,000:

```
select DEPT_ID_DEP, DEP_NAME from departments  
where DEPT_ID_DEP IN  
( select DEP_ID from employees  
where SALARY > 70000 ) ;
```

Result:

DEPT_ID_DEP	DEP_NAME
5	Software Group

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

## Accessing multiple tables with Implicit Join

Specify 2 tables in the FROM clause:

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

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

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

Result:

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

Use additional operands to limit the result set:

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

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

Use shorter aliases for table names:

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

Result:

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

To see the department name for each employee:

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

Result:

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

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

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

Result:

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

### Hands-on Lab: Working with Multiple Tables

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

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

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

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

### Exercise 1: Accessing Multiple Tables with Sub-Queries

1. Problem:

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

Solution

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

Output

Result set 1										
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 In, 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										
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. Architect	60000.00	100000.00	100
Sr.Software Dev	60000.00	80000.00	200
Lead Architect	70000.00	100000.00	600

## 4. Problem:

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

### Solution

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

## Output

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

### 5. Problem:

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

## Solution

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

## Output

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

## Exercise 2: Accessing Multiple Tables with Implicit Joins

### 1. Problem:

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

## Solution

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

## Output

Result set 1														
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 In, 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	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														
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 In, 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														
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	1991-01-04	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;
```

#### 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

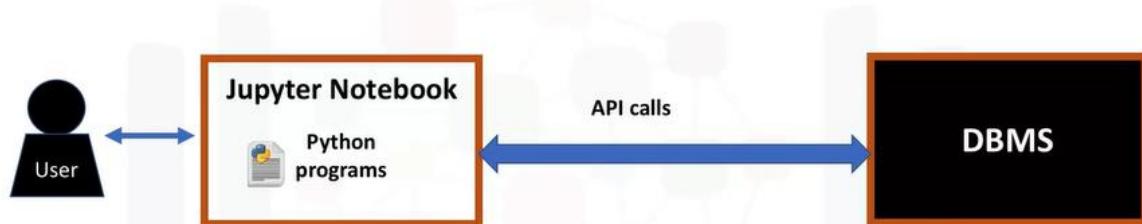
## Solution Script

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

- [MultipleTables\\_Solution\\_Script.sql](#)

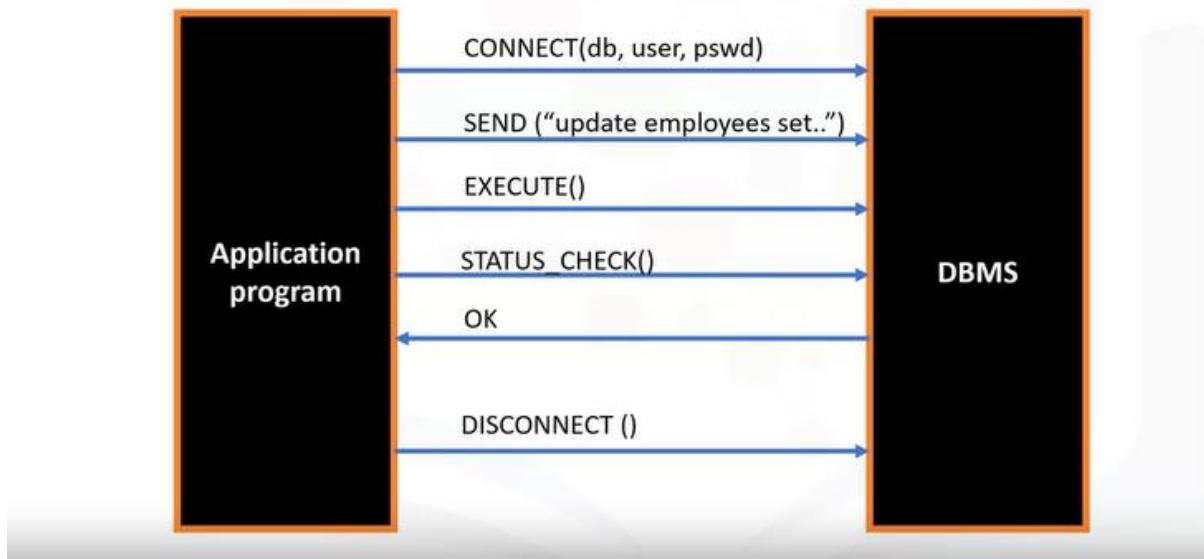
## How to Access Databases Using Python

### Accessing databases using Python



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

## What is a SQL API?

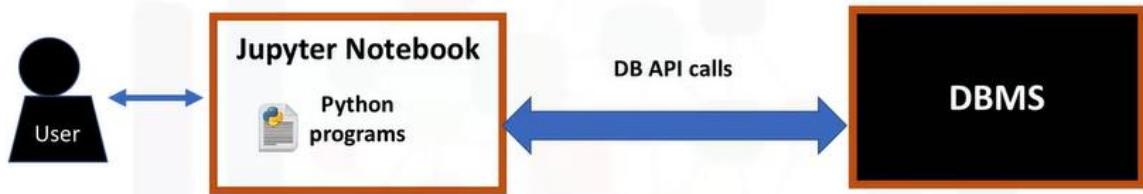


APIs used by popular SQL-based DBMS systems

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

## Writing code using DB-API

## What is a DB-API?



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

## Benefits of using DB-API

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

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

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

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

## Concepts of the Python DB API

### Connection Objects

- Database connections
- Manage transactions

### Cursor Objects

- Database Queries
- Scroll through result set
- Retrieve results

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.

## What are Connection methods?

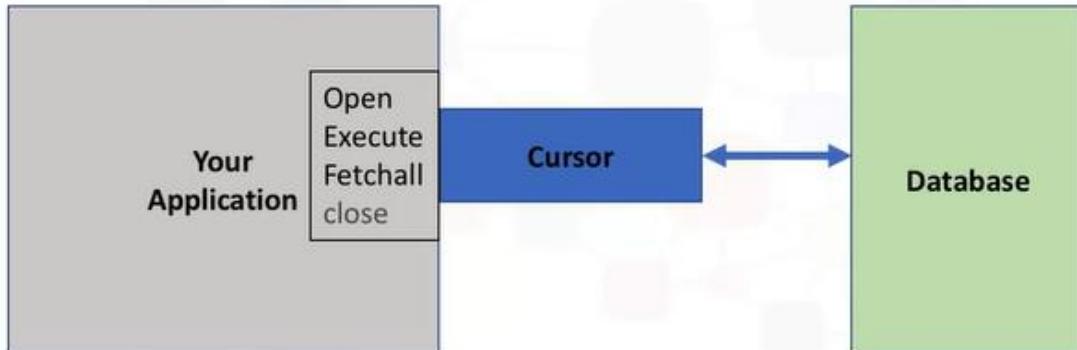
- `.cursor()`
- `.commit()`
- `.rollback()`
- `.close()`

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.

## What are cursor methods?

- `.callproc()`
- `.execute()`
- `.executemany()`
- `.fetchone()`
- `.fetchmany()`
- `.fetchall()`
- `.nextset()`
- `.arraysize()`
- `.close()`

# What is a database cursor?



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.

## Writing code using DB-API

<pre> from dbmodule import connect #Create connection object Connection = connect('databasename', 'username', 'pswd') </pre>	<b>#Run Queries</b> <pre> Cursor.execute('select * from mytable') Results=cursor.fetchall() </pre>
<b>#Create a cursor object</b> <pre> Cursor=connection.cursor() </pre>	<b>#Free resources</b> <pre> Cursor.close() Connection.close() </pre>

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

## Connecting to a database using ibm\_db API

### 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"
```

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

# Create a database connection

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

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

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

Connected!

# Close the database connection

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

Out[8]: True

## Create credentials to access your database instance

### Exercise

Database credentials are required to connect from remote applications like Jupyter notebooks which are used in the labs and assignment in the last two weeks of the course.

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

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

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

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

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

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

The screenshot shows a sidebar menu with the following items:

- Devices (0)
- VPC infrastructure (0)
- Clusters (0)
- Container Registry (0)
- Satellite (0)
- Cloud Foundry apps (0)
- Cloud Foundry services (0)
- Services and software (1)

The "Services and software" item is expanded, revealing a list of services:

- Db2-xm (highlighted with a red box)
- Storage (0)
- Network (0)
- Functions namespaces (0)
- Apps (0)
- Developer tools (0)

On the right, there is a "Default" label.

2. Click on Service Credentials in the left menu

The screenshot shows the service details for "Db2-xm". The top bar includes "Resource list /", the service name "Db2-xm", an "Active" status indicator, and a "Add tags" button.

The main area has two tabs: "Manage" (selected) and "Getting started".

The "Manage" tab contains three buttons: "Getting started", "Service credentials" (highlighted with a red box), and "Connections".

The "Getting started" tab contains the following text:

Where can I find my credentials?  
Get your username and password by clicking the "Service Credential" "New Credentials".

At the bottom of the "Getting started" tab are two buttons: "Go to UI" and "Getting started docs".

3. Click on the button to create New credential

A modal dialog box is displayed with the title "New credential" and a "+" button in the bottom right corner.

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

## Create credential

X

Name:

Service credentials-3

Role: ⓘ

Manager

[Advanced options ▾](#)

Cancel

Add

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

```
{
  "hostname": "764264db-9824-4b7c-82df-40d1b13897c2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud",
  "port": 32536
}
```

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

**"port": 32536**

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

```
"composed": [
  "db2 -u nvg16884 -p SC8jmSzwbWQ22x9K --ssl --sslCAFile 1dd14d0c-1b52-4f63-a606-53ecba28771d --authenticationDatabase admin -h host 764264db-9824-4b7c-82df-40d1b13897c2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:32536"
],
"environment": {},
```

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

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

```
"password": "SC8jmSzwbWQ22x9K",
"username": "nvg16884"
```

**"password": "SC8jmSzwbWQ22x9K",**

**"username": "nvg16884"**

```
Service credentials:1 2021-09-27 11:07 AM

{
  "connection": {
    "cli": {
      "arguments": [
        [
          "-u",
          "nvg16884",
          "-p",
          "SC8jM5zmbWQ22x9K",
          "-s",
          "-scafile",
          "1dd14d0c-1b52-4f63-a606-53ecba28771d",
          "-authenticationDatabase",
          "admin",
          "-host",
          "764264db-9824-4b7c-82df-40d1b13897c2.es2io90108qkblod8lcg.firebaseio.googleapis.com:32536"
        ],
        {
          "bin": "db2",
          "certificate": {
            "certificate_base64": "LS0tLS1CRUJDbTBDRVJUSUZQ0FURS0tLS0tCk1JSURVENDQWxZ0F3SUJBZ01V3dvcMc9va09CUENRjFWeJxJvGHKRw9ubDBvd0RRWUpLb1pJaHzJTkFRUwQ1Fbd0hqrRN
Qm9HQTFRVdRTVUp0SUv0C21zVntRVJ0zEdgV1YTMxjkF1Rncwe1UQTRNRF3TwpVmwpNalpHKnwke1EQTRNRF3TwpMo1qwmFNQjR4SERBYUJnT1ZCQU1NRTBsQ1RTQkRzKxwNCRV1YUmhZbuZ6C1pYTx
nZ0V9PQ1F0N7QdTSW1hNE12d0nRtbh01CQ0ZNG05Qd0eXZUMW1rN4MVBlJ0N7RSt1NLRK2zidRuQ0SuJy09V23y4Y1g1T61GxQkzY1F0G9wVsxiZQ30ExoaxRZQ
pySm112U11z1F4WTFM0c3BqGRFVEZKwHeScnJhMGU2VmM4W42711JL0ZhsTlZG5MuFtQWP9hWyadM2ChndTPOFxRwT1FtZnMTPGeU0yDR1ja1kccK14RaK4v9XMyVdVfHMNgwZxLUZVU
CeRzUmJ3VkyS
Vc3aMkbGpN3R1N3h2Vp0U0uYh1E1Yd1oTBR5RVRZH1ESVYUEZGRD8BHylz1A2o3M29dpvJu3V39uHr9HOTJNw825kdzTnpKNWp0pFBNT3V2a2zUVHIN1D1BaUVNxpQdUpV
VzNOY9R0qCmV2a2zBZ01CQUPHa1VqQ1JMjQhFTVZcErnDfCQ1R2RzZRU5MhFVbwZnQ93MmxOcmCmD12bURBzJnT1YKkNNRUe0Vgn1R2RzZRU5MhFVbwZnQ93MmxOcmCmD12bURBzJnT1Z1Uk1C0QWY
```

You need to scroll down to get the credentials details.

```
    "plset"
  ],
  "database": "bludb",
  "host_ros": [
    "764264db-9824-4b7c-82df-40d1b13897c2.bs2io90108kqb1od8lcg.databases.appdomain.cloud:32761"
  ],
  "hosts": [
    {
      "hostname": "764264db-9824-4b7c-82df-40d1b13897c2.bs2io90108kqb1od8lcg.databases.appdomain.cloud",
      "port": 32536
    }
  ]
}
```

## Hands-on Lab: Connecting to a database instance

This Lab will demonstrate how to create a database connection to an instance of IBM Db2 on Cloud. You will be using Jupyter notebooks to complete the labs. Click [HERE](#) to download the lab notebook (.ipynb). Local file can be downloaded from here → [Python File](#)

Assignment can be seen from here → [Python File](#)

## Creating tables, loading data and querying data

We will be using DB2 as the database.

# Connect to the database

```

import ibm_db

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"

#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!

```

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

## ibm\_db.exec\_immediate()

The parameters for the function are:

- Connection
- Statement
- Options

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

## Python code to create a table

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

## Python code to insert data into the table

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

### Insert more rows to the table

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

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

# Python code to query data

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

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

Confirm the output on Db2 on Cloud console

The screenshot shows a database table named 'TRUCKS' with the following columns and data:

	SERIAL_NO VARCHAR(20)	MODEL VARCHAR(20)	MANUFACTURER VARCHAR(20)	ENGINE_SIZE VARCHAR(20)	TRUCK_CLASS VARCHAR(20)
1	A1234	Lonestar	International Trucks	Cummins ISX15	Class 8
2	B5432	Volvo VN	Volvo Trucks	Volvo D11	Heavy Duty Class 8
3	C5674	Kenworth W900	Kenworth Truck Co	Caterpillar C9	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
```

The screenshot shows a pandas DataFrame with the same structure and data as the Db2 table:

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

**Hands-on Lab: Creating tables, inserting and querying Data**  
[Lab Notebook](#)

Click [HERE](#) to download the lab notebook (.ipynb)

## Introducing SQL Magic

### Objective

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

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

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

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

Their usage is of the format:

### **%magicname arguments**

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

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

**%sql select \* from tablename**

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

### **Hands-on Tutorial: Accessing Databases with SQL magic**

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