Course 3 - Welcome to SQL: A Practical Introduction for Querying Databases!

Getting Started with SQL

Week 1

Module 1: - Basic SQL

1] Course Introduction

Who uses SQL?

- Data Scientists
- Data Analysts
- Data Engineers
- Data Warehouse Specialists
- Business Intelligence Analysts
- Data Architects, and
- Database Administrators

All rely on their SQL skills to query data or perform analysis.

Why SQL?

About SQL –

- SQL is a powerful language for communicating with databases.
- Whether it's just a few rows in a single table, or billions of rows in many tables across servers.

Advantages to learning SQL –

- SQL will boost your profile as a data professional.
- SQL is one of the most in-demand skills.
- SQL will give you an understanding of many database types.

What you will Learn –

- Basics of SQL and relational databases
- Perform queries using SELECT, INSERT, UPDATE and DELETE statements
- Create a databases instance, create tables, and load data
- Filter, order, aggregate your results
- Write nested queries & join data in many tables
- Analyze real world data set with SQL

2] Introduction to Databases

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 database?

- A repository of data
- Program that stores data
- Provides the functionality for adding, modifying and querying that data
- Different kinds of databases store data indifferent 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”)

DBMS: -

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

What is RDBMS?

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

Basic SQL Commands: -

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

<u>Summary: -</u>

- You can now describe:
  - SQL
  - Data
  - Database
  - Relational Databases
  - RDBMS
- 5 basic SQL commands:

- Create, insert, select, update, delete

3] SELECT Statement

Retrieving rows from a table: -

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

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

Select * from <tablename>
```

Using the SELECT Statement: -

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

Example: select * from Book

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

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

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

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

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

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

Retrieving a subset of the columns: -

- You can retrieve just the columns you want
- The order of the columns displayed always matches the order in the SELECT statement.

SELECT <column 1>, <column 2> from Book

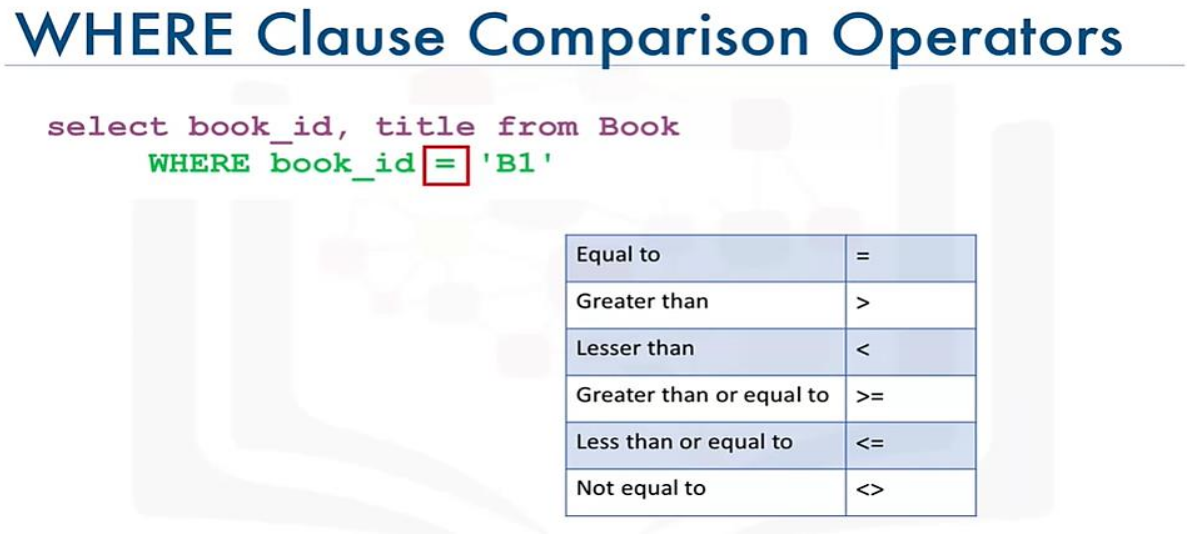
```
db2 => select book_id, title from Book

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

  4 record(s) selected.
```

Restricting the result Set: WHERE Clause: -

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

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

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

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

  1 record(s) selected
```



Summary: -

Now you can:

- Retrieve data from a relational table.
- Define the use of a paradise.
- Identify the syntax of SELECT statement using the WHERE clause.

List the comparison operators supported by a RDBMS

SELECT statement examples

The general syntax of SELECT statements is:

select COLUMN1, COLUMN2, … from TABLE1 ;
To retrieve all columns from the COUNTRY table we could use "*" instead of specifying individual column names:

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

select * from COUNTRY where ID < 5 ;
In case of character-based columns the values of the predicates in the where clause need to be enclosed in single quotes. To retrieve the data for the country with country code "CA" we would issue:

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

4] COUNT, DISTINCT, LIMIT

1. 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
```

Example: -

Rows in the MEDALS table where Country is Canada:

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

Result:

```
            1
--------------
           29
```

2. DISTINCT: –

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

Retrieve unique values in a column:

```
select DISTINCT columnname from tablename
```

Example: -

List of unique countries that received GOLD medals:

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

Result:

```
              1
---------------
             21
```

3. 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
```

Example: -

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

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

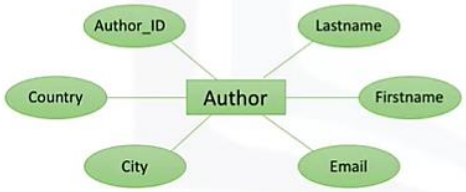
Result:

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

5] INSERT Statement

Adding row to a table: -

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



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

Using the INSERT Statement: - Syntax with Example

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

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

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

Inserting multiple rows: -

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

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

Summary: -

Now you can:

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

6] 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: -

Syntax:

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

Example:

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

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

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

Deleting rows from a table – DELETE Statement: -

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

Syntax:

```
DELETE FROM [TableName]
<WHERE [Condition]>
```

Example:

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

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

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

Summary: -

Now you can:

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

Summary & Highlights:-

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

Week 2

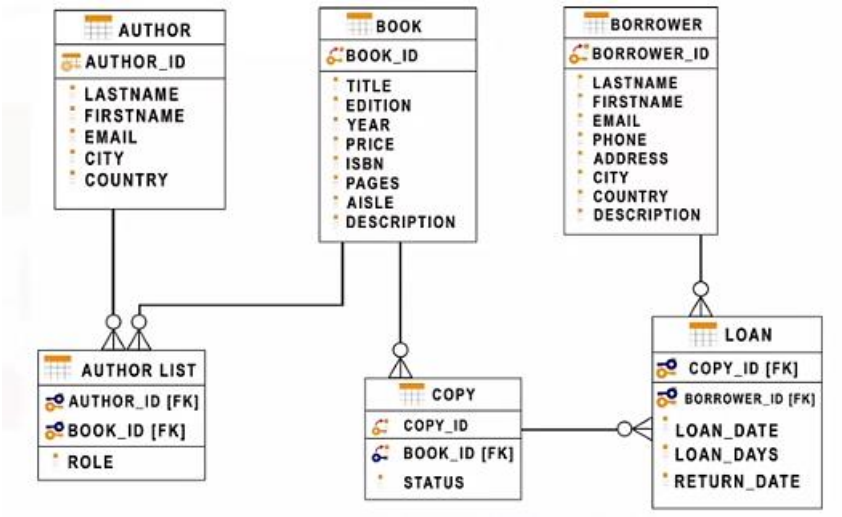
Introduction to Relational Databases and Tables

Module 1: - Introduction to Relational Databases and Tables

1] Relational Database Concepts

Relational Model: -

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



Logical data independence – physical data independence-physical storage independence

Entity-Relational Model: -

- Used as a tool to design relational databases.

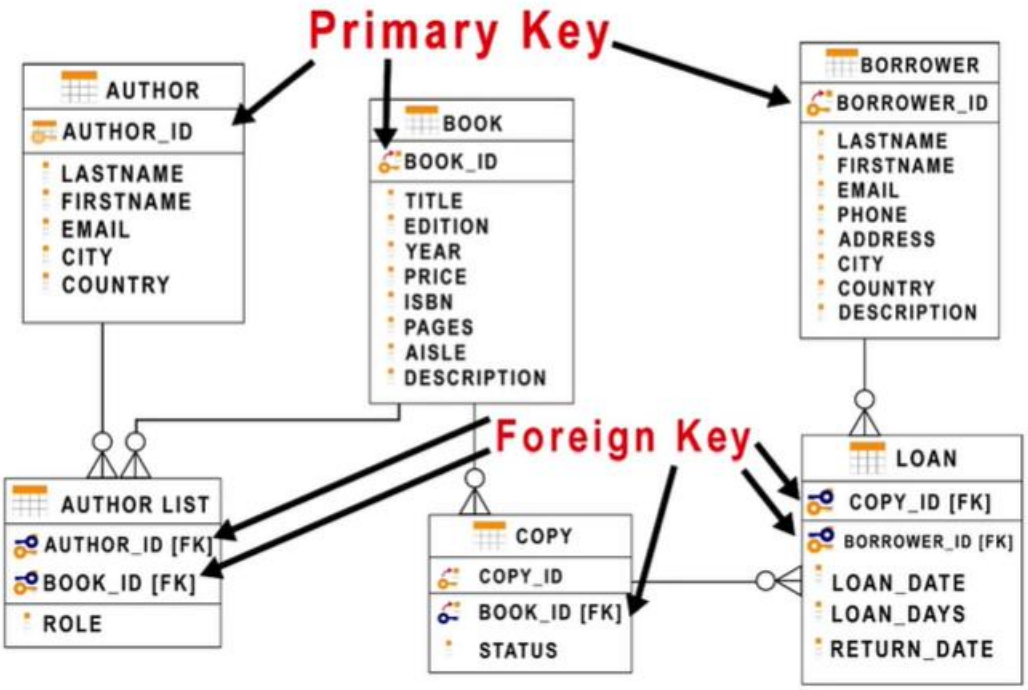
Mapping Entity Diagrams to Tables: -

- Entity become tables
- Attributes get translated into columns

Table: Book

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

Primary Keys and Foreign Keys: -



Summary: -

Now you know:

- The key advantage of the relational model is logical and physical data independence and storage independence.
- Entities are independent objects which can have multiple characteristics called attributes.
- When mapping to a relational database, entities are represented as tables and attributes map to columns.

- Common data types include characters such as car and VAR char, numbers such as integer and decimal, and timestamps including date and time.
- A primary key uniquely identifies a specific row in a table and prevents duplication of data.

2] 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

Types of SQL Statements – DML: -

- DML (Data Manipulation Language) statements:
  - Read and modify data
  - CRUD operations (Create, Read, Update & Delete rows)
- Common DML:
  - INSERT
  - SELECT
  - UPDATE
  - DELETE

Summary: -

Now you know that:

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

3] CREATE TABLE Statement

CREATE table: -

Syntax –

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

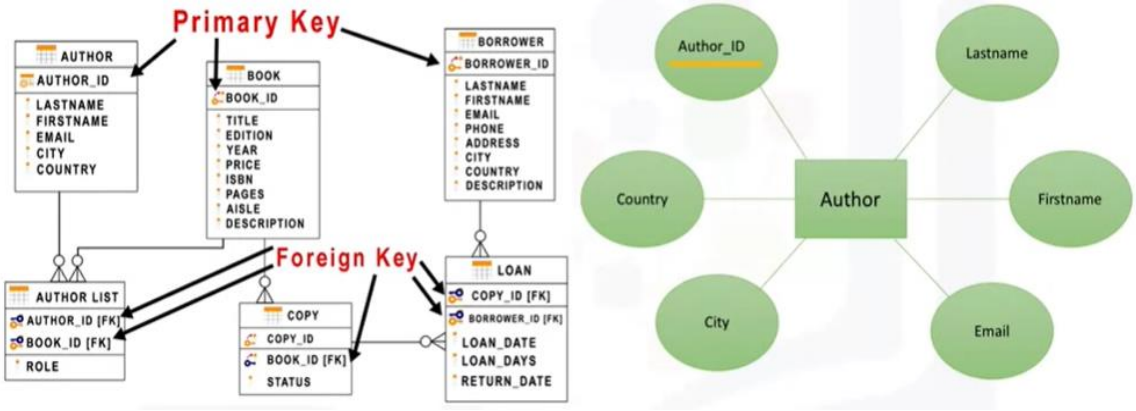
Example –

Create a table for Canadian provinces

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

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

Create a table



- **Primary Key:** Uniquely Identifies each Row in a Table.

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

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

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

Note:

- Author_ID is the Primary Key. This constraint prevents duplicate values in the table.
- Also, 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: -

- CREATE is a DDL statement for creating Entities or tables in a database.
- The CREATE TABLE statement includes definition of attributes of columns in the table, including:
  - Names of columns
  - Datatypes of columns and
  - Constraint (e.g., Primary Key)

4] ALTER, DROP, and Truncate tables

1. ALTER statement: -

ALTER TABLE…….ADD COLUMN: -

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

Syntax –

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

Example –

- To add a telephone number column to the AUTHOR table in the library database to store the author's telephone number, use the following statement:

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

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

ALTER TABLE…….ALTER COLUMN: -

Syntax –

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

Example –

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

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

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

ALTER TABLE…….DROP COLUMN: -

- To delete a column from a table.

Syntax with example –

- Changing a column from the CHAR data type to a numeric data type will not work if the column already contains non-numeric data. If you try to do this, you will see an error message in the notification log and the statement will not run. If your spec changes and you no longer need this extra column, you can again use the ALTER TABLE statement, this time with the DROP COLUMN clause, to remove the column as shown:

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

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

2. DROP statement: -

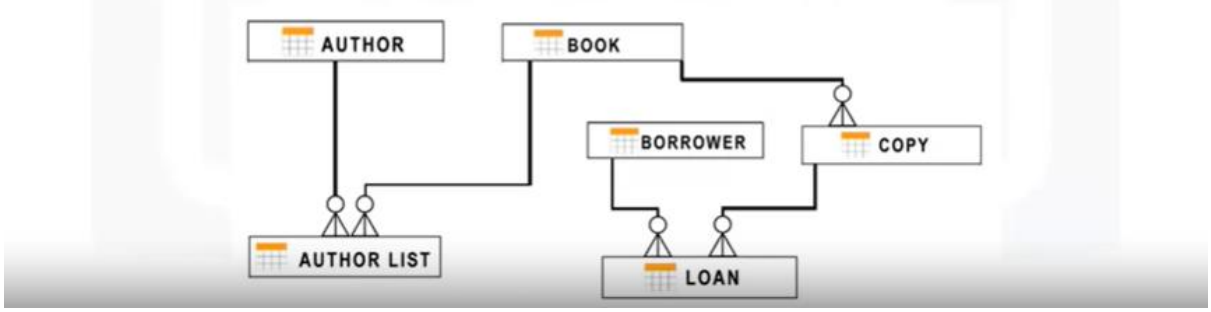
- To delete a table from a database.
- If you delete a table that contains data, by default the data will be deleted alongside the table.

Syntax –

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

Example –

```
DROP TABLE author;
```



3. TRUNCATE statement: -

- To delete all of the rows in a table.

Syntax –

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

- IMMEDIATE specifies to process the statement immediately and that it cannot be undone.

Example –

```
TRUNCATE TABLE author
  IMMEDIATE;
```

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

Summary: -

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

Examples to ALTER and TRUNCATE tables using MySQL

ALTER TABLE

ALTER TABLE statements used to add or remove columns from a table, to modify the data type of columns, to add or remove keys, and to add or remove constraints. The syntax of the ALTER TABLE statement is:

ADD COLUMN syntax

```
ALTER TABLE table_name

ADD COLUMN column_name data_type column_constraint;
```

For example, to add a **telephone_number** column to the **author** table in the **library** database, the statement will be written as:

```
ALTER TABLE author

ADD COLUMN telephone_number BIGINT;
```

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

ALTER TABLE column syntax

```
ALTER TABLE table_name CHANGE column_name

column_name VARCHAR(20);
```

For example, using a **numeric** data type for **telephone_number** means that you cannot include **parentheses**, **plus signs**, or **dashes as part of the number**.

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

You can change the column to use the **CHAR** data type to overcome this. To modify the data type, the statement will be written as:

```
ALTER TABLE author CHANGE telephone_number

telephone_number CHAR(20);
```

TRUNCATE Table

TRUNCATE TABLE statement are used to delete all of the rows in a table. The syntax of the statement is:

```
TRUNCATE TABLE table_name;
```

So, to truncate the author's table, the statement will be written as:

```
TRUNCATE TABLE author;
```

author_id	lastname	firstname	email	city	country

TRUCATE statement will just delete the rows and not the table.

Examples to CREATE and DROP tables

Here we will look at some examples to create and drop tables.
In the previous video we saw the general syntax to create a table:
create table TABLENAME (
COLUMN1 datatype,
COLUMN2 datatype,
COLUMN3 datatype,
…
);
Therefore, to create a table called TEST with two columns - ID of type integer, and Name of type varchar, we could create it using the following SQL statement:

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

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

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

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

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

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

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

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

WARNING: before dropping a table ensure that it doesn't contain important data that can't be recovered easily.
Note that if the table does not already exist and you try to drop it, you will see an error like **XXX.YYY is an undefined name**. You can ignore this error as long as the subsequent CREATE statement executed successfully.
In the lab later in this module you will practice creating tables and other SQL statements hands-on.

Module 2: - Optional: Hands-on Labs with Db2

1] How to create a Database instance on Cloud

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: -

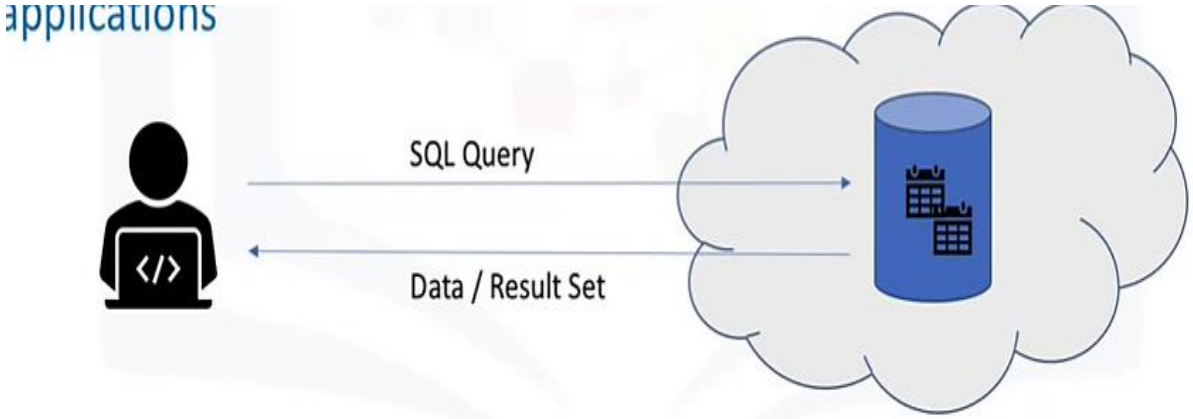
- IBM Db2
- Databases for PostgreSQL
- Oracle Database Cloud Service
- Microsoft Azure SQL Database
- Amazon Relational Database Services (RDS)

Available as:

- VMs or Managed Service
- Single or Multi-tenant

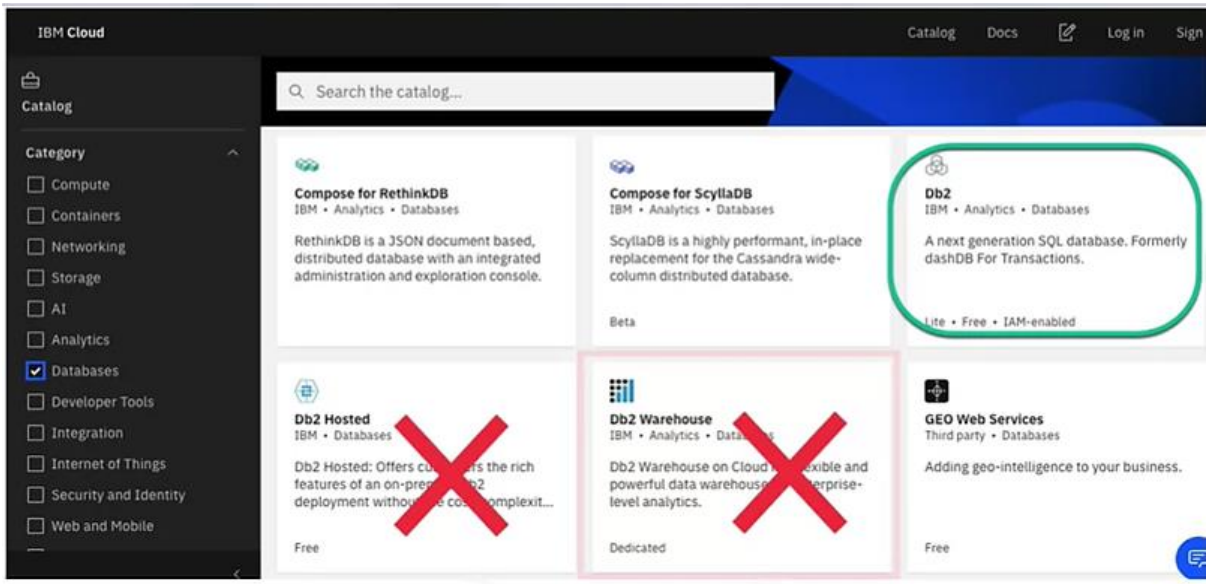
Database service instances: -

- 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.

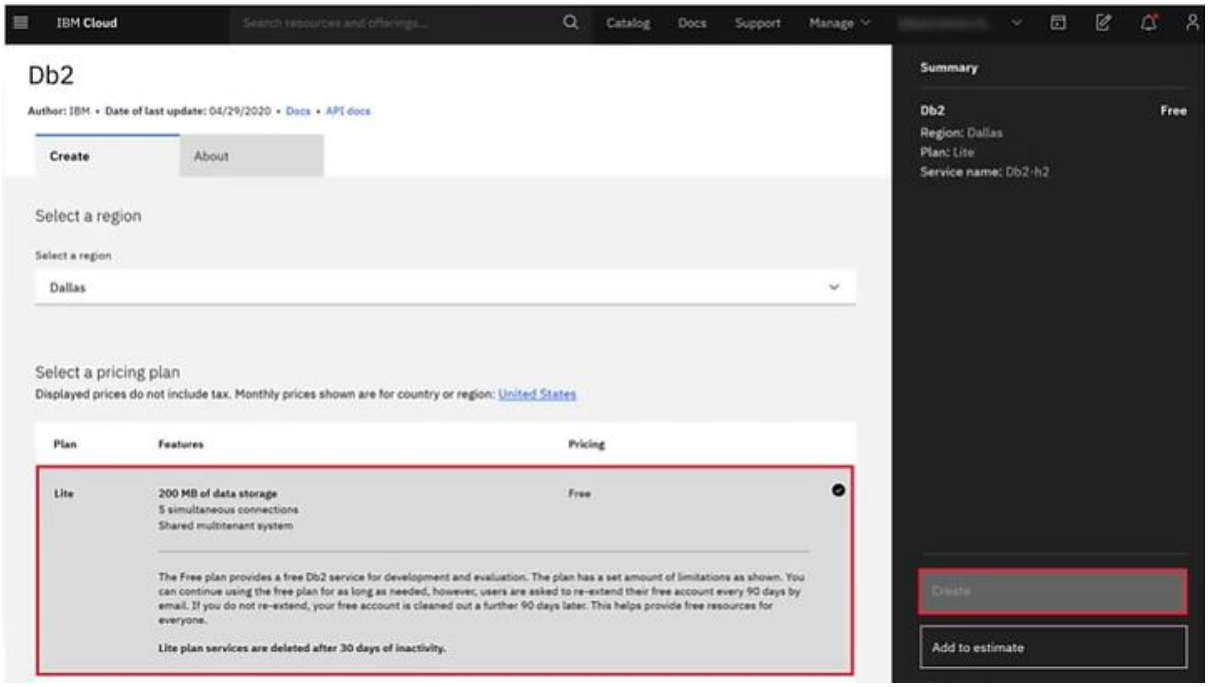


Creating a Database instance on IBM Db2 on Cloud -

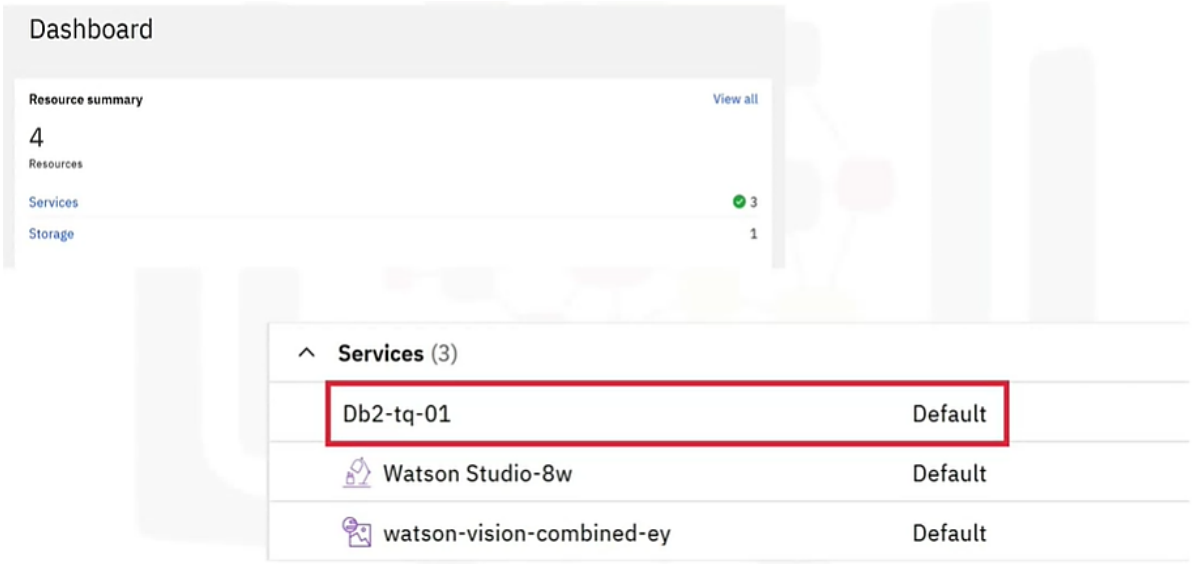
Deploy an instance of Db2 on Cloud Service –



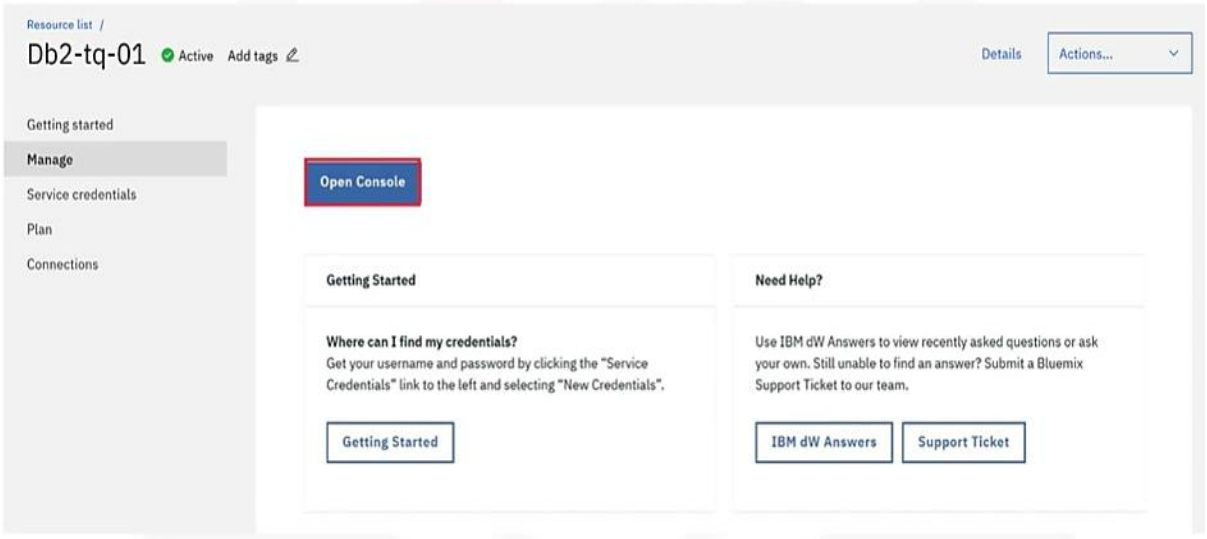
Create a new Service –



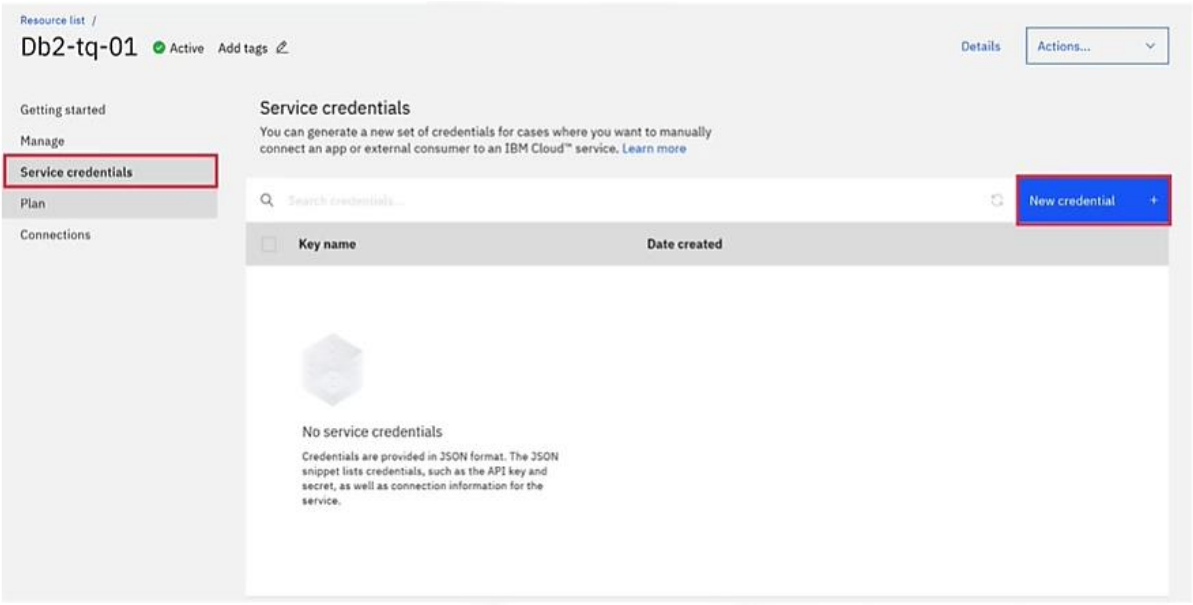
View the newly created service –



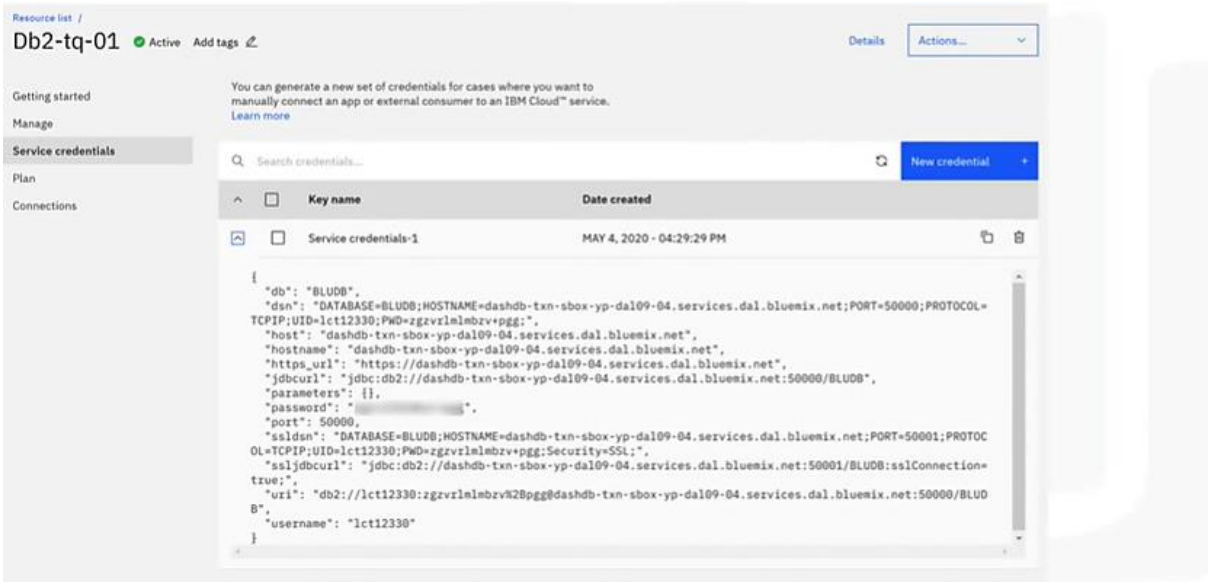
Manage the database instance –

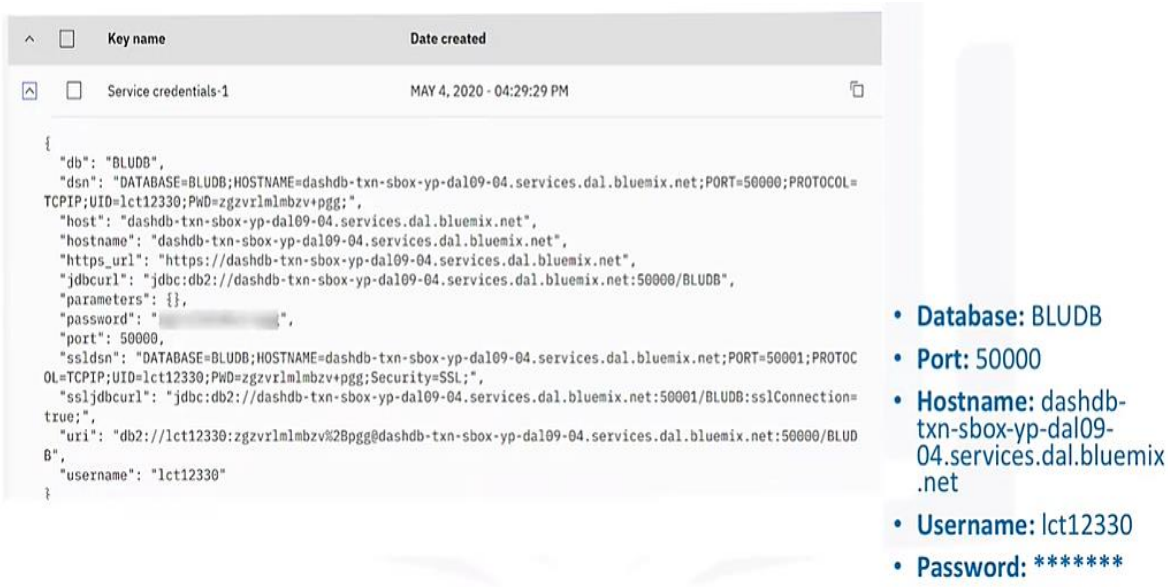


Create new service credentials –



Service credentials –





Summary & Highlights:-

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

Week 3

Intermediate SQL

Module 1: - Refining your Results

1] Using String Patterns and Ranges

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.
```

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

Syntax with Example:

- WHERE <columnname> LIKE <string pattern>

```
WHERE firstname LIKE R%
```

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

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

Retrieving rows – using a Range: -

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

Title                                Pages
-----------------------------------  -----
Database Fundamentals                300
Getting started with DB2 App Dev     298
     2 record(s) selected.
```

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

Title                                Pages
-----------------------------------  -----
Database Fundamentals                300
Getting started with DB2 App Dev     298
     2 record(s) selected.
```

Retrieving rows – using a Set of Values: -

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

Firstname       Lastname        Country
--------------  --------------  -------
Xiqiang         Ji              AU
Juliano         Martins         BR
     2 record(s) selected.
```

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

Firstname       Lastname        Country
--------------  --------------  -------
Xiqiang         Ji              AU
Juliano         Martins         BR
     2 record(s) selected.
```

Summary: -

Now you can describe how to simplify a SELECT statement by using:

- String Patterns
- Ranges, or
- Sets of values

2] Sorting Result Sets

Sorting the Result Set: -

```
db2 => select * from Book

BOOK_ID  TITLE                          EDITION  YEAR  PRICE  ISBN              PAGES  AISLE   DESCRIPTION
B1       Getting started with DB2 Express-C  1  2010  24.99  978-0-98006283-1-1  300  DB-A02  Teaches you the fundamentals
B2       Database Fundamentals          1  2010  24.99  978-0-98666283-5-1  280  DB-A01  Teaches you the essentials of
B3       Getting started with DB2 App Dev  1  2011  35.99  978-0-98086283-4-1  345  DB-A03  Teaches you the essentials of
B4       Getting started with WAS CE    1  2010  49.99  978-0-98946283-3-1  458  DB-A04  Teaches you the essentials of

  4 record(s) selected.
```

```
db2 => select title from Book

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

  4 record(s) selected.
```

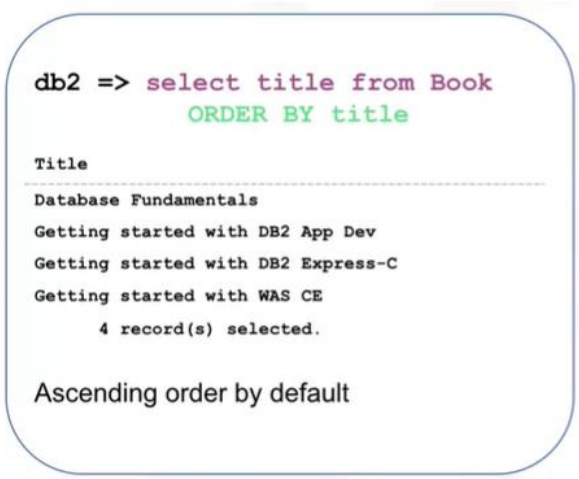
Using the ORDER BY clause: -

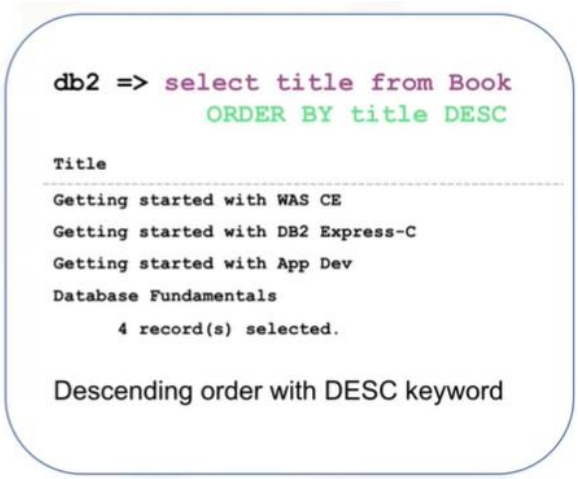
```
db2 => select title from Book

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

  4 record(s) selected.
```

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

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

  4 record(s) selected.
```

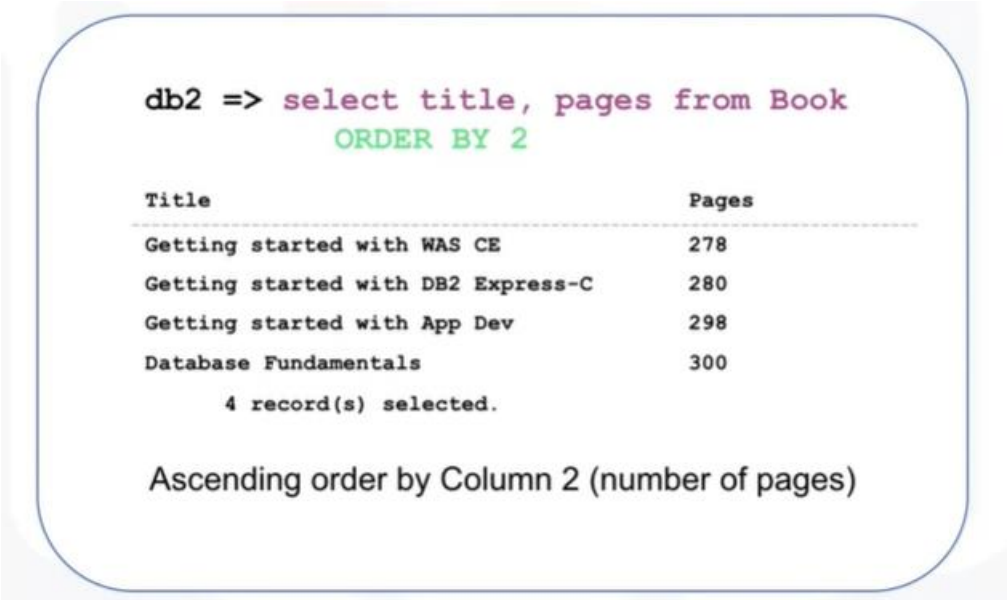
By default the result set is sorted in ascending order

ORDER BY clause – Descending order: -





Specifying Column Sequence Number: -



Summary: -

- 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.

3] Grouping Result Sets

Eliminating Duplicates – DISTINCT clause: -

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

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

  20 record(s) selected.
```

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

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

  6 record(s) selected.
```

GROUP BY clause: -

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

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

  20 record(s) selected.
```

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

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

  6 record(s) selected.
```

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

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

      20 record(s) selected.
```

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

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

      6 record(s) selected.
```

Restricting the Result Set - HAVING clause: -

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

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

      6 record(s) selected.
```

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

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

      6 record(s) selected.
```

<u>Summary: -</u>

Now you can:

- Eliminate duplicates from a result set
- Describe how to further restrict a result set

Summary & Highlights:-

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

Module 2: - Functions, Multiple Tables, and Sub-queries

1] Built-in Database Functions

Built-in Functions: -

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

PETRESCUE TABLE

PETRESCUE

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

Aggregate or Column Functions: -

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

1. **SUM –**

    - **SUM Function:** Add up all the values in a column

```
SUM(COLUMN_NAME)
```

Example:

Example 1: Add all values in the COST column:

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

Example 1: Result:

```
1
1718.24
```

Column Alias:

Example:

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
```

2. **MIN, MAX –**

- **MIN():** Return the MINIMUM value
- **MAX():** Return the MAXIMUM value

Example:

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
```

3. **AVG –**

- **AVG():** return the average value

Example:

Example 4. Specify the Average value of COST:

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

Example 4. Results:

```
1
190.91555555555555555555555555
```

- Mathematical operations can be performed between columns.

Example:

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

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

Example 5. Results:

```
1
127.27000000000000000000000000
```

SCALER and STRING Functions: -

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

Example:

Example 6: Round UP or

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

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

Example 6. Results:

Example 7. Results:

```
1
3
3
3
6
3
```

UCASE. LCASE: -

Example:

Example 8: Retrieve ANIMAL values in UPPERCASE:

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

Example 8: Results:

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

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

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
```

2] Date and Time Built-in Functions

Date, Time Functions: -

- Most databases contain special datatypes for dates and times.

DATE:	YYYYMMDD
TIME:	HHMMSS
TIMESTAMP:	YYYYXXDDHHMMSSZZZZZZ

Date/Time functions: -

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

Example:

Example 11: Extract the DAY portion from a date:

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

Example 11: Results:

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

29

11

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

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

Example 12: Results:

```
1
```

Date/Time Arithmetic: -

Example:

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

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

Example 13: Results:

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

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

Special Registers: -

```
CURRENT_DATE, CURRENT_TIME
```

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

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

Example 14: Sample result (format YMMDD):

```
10921
```

3] Sub-Queries and Nested Selects

Sub-query: -

- A query inside another query.

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

EMPLOYEES

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

Why use sub-queries: -

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

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

This query will result in error:

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

Sub-queries to evaluate Aggregate functions: -

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

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

Sub-queries in list of columns: -

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

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

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

Sub-queries in list of columns: -

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

Example:

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

Summary: -

In this video you have learned:

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

4] 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 );
```

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

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

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.

Use additional operands to limit the result set:

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

Use shorter aliases for table names:

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

Query:

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

Result:

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

To see the department name for each employee:

Query:

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

Result:

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

Query:

```
select E.EMP_ID, D.DEPT_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

Summary & Highlights: -

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

Week 4

Working with real-world data sets, Final Project & Exam

Module 1: - Assignment Preparation: working with real-world data sets and built-in SQL functions

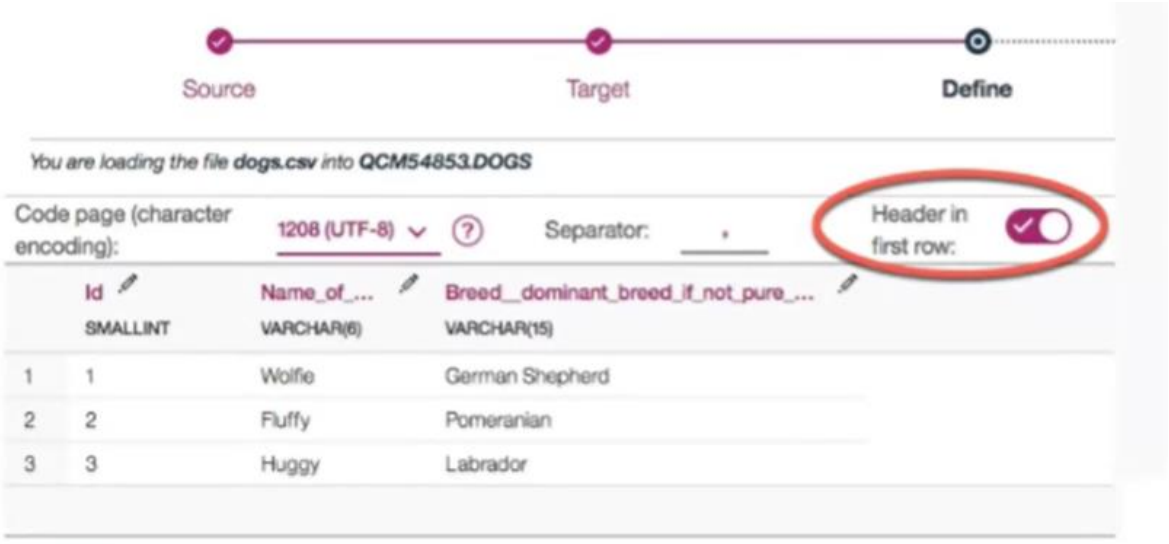
1] Working with Real World Datasets

- Many real data sets are .CSV files
- .CSV: COMMA SEPARATED VALUES
- Example: DOGS.CSV

```
Id,Name of Dog,Breed (dominant breed if not pure breed)
1,Wolfie,German Shepherd
2,Fluffy,Pomeranian
3,Huggy,Labrador
```

Column names in first row –

When header row in CSV file contains column names:



Querying column names with mixed (upper and lower) case –

Retrieve Id column from DOGS table. Try:

```
select id from DOGS
```

If you run this query, you will get this error:

```
Error: "ID" is not valid in the context where it is
used.. SQLCODE=-206, SQLSTATE=42703, DRIVER=4.22.36
```

Use double quotes to specify mixed-case column names:

```
select "Id" from DOGS
```

Querying column names with spaces and special characters –

By default, spaces are mapped to underscores:

	A
1	Name of Dog
2	

```
Name_of_Dog
```

Other characters may also get mapped to underscores:

```
select "Id", "Name_of_Dog",
"Breed__dominant_breed_if_not_pure_breed_"
from dogs
```

Breed (dominant breed if not pure breed)

Using quotes in Jupyter notebooks –

First assign queries to variables:

```
selectQuery = 'select "Id" from dogs'
```

Use a backslash \ as the escape character in cases where the query contains single quotes:

```
selectQuery = 'select * from dogs
      where "Name_of_Dog"=\'Huggy\' '
```

Splitting queries to multiple lines in Jupyter –

Use backslash "\" to split the query into multiple lines:

```
%sql select "Id", "Name_of_Dog", \
    from dogs \
    where "Name_of_Dog"='Huggy'
```

Or use %%sql in the first row of the cell in the notebook:

```
%%sql
select "Id", "Name_of_Dog",
    from dogs
    where "Name_of_Dog"='Huggy'
```

Restricting the # of rows retrieved –

To get a sample or look at a small set of rows, limit the result set by using the LIMIT clause:

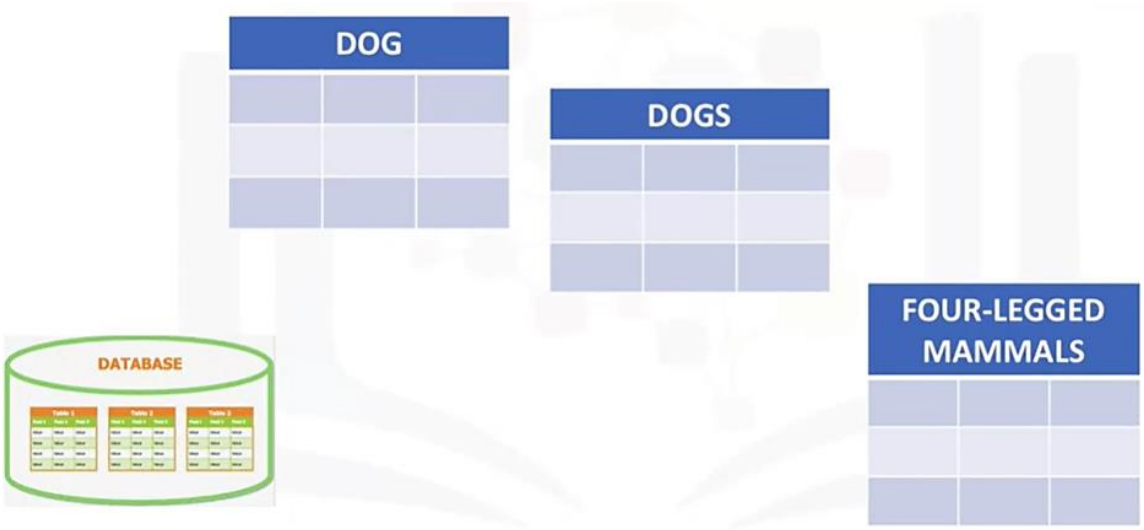

```
select * from census_data LIMIT 3
```

Summary: -

- CSV is a common data format for loading tables
- CSV files may contain a first header row that needs to be treated differently from data rows.
- Many RDBMSes like Db2 by default map mixed case column names to uppercase and spaces and special characters to underscores.
- Data in a table can be easily sampled using the LIMIT clause in a SELECT query to retrieve just a few rows of data.

2] Getting Table and Column Details

Getting a list of tables in the database –



DB2
SYSCAT.TABLES
SQL Server
INFORMATION_SCHEMA.TABLES
Oracle
ALL_TABLES or USER_TABLES

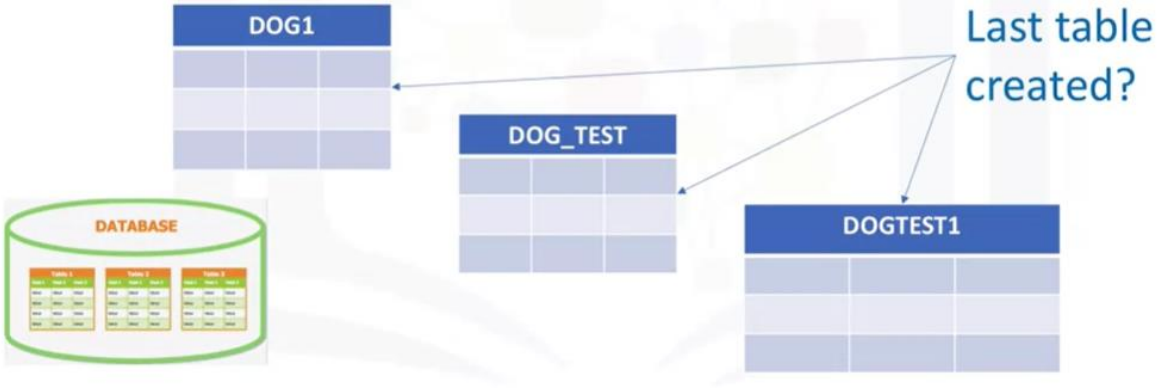
Query system catalog to get a list of tables and their properties:

```
select * from syscat.tables
```


```
select TABSCHEMA, TABNAME, CREATE_TIME
    from syscat.tables
    where tabschema= 'ABC12345'
```

Getting Table Properties:

```
select * from syscat.tables
```




```
select TABSCHEMA, TABNAME, CREATE_TIME
     from syscat.tables
     where tabschema='LCT12330'
```

```
%sql select TABSCHEMA, TABNAME, CREATE_TIME from SYSCAT.TABLES where TABSCHEMA='LCT12330'
 * ibm_db_sa://lct12330:***@dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net:50000/BLUDB
Done.
```

tabschema	tabname	create_time
LCT12330	PETRESCUE	2020-05-05 22:02:25.169368
LCT12330	BOOK	2020-02-20 22:37:50.351829
LCT12330	MEDALS	2020-02-22 02:50:54.841324
LCT12330	EMPLOYEES	2020-02-18 20:05:38.328981
LCT12330	JOB_HISTORY	2020-02-18 20:05:38.521203
LCT12330	JOBS	2020-02-18 20:05:38.700230
LCT12330	LOCATIONS	2020-02-18 20:05:39.050699
LCT12330	SCHOOLS	2020-04-16 22:06:58.174131
LCT12330	DEPARTMENTS	2020-04-15 20:01:16.429000
LCT12330	MCDONALDSNUTRITION	2020-04-23 16:40:35.205237

<u>Getting a list of columns in the database</u> –

To obtain the column names query syscat.columns:

```
select * from syscat.columns
        where tabname = 'DOGS'
```

To obtain specific column properties:

```
select distinct(name), coltype, length
       from sysibm.syscolumns
       where tbname = 'DOGS'
```

Column info for a real table –

```
In [12]: %sql select distinct(name), coltype, length \
              from sysibm.syscolumns where tbname = 'CHICAGO_CRIME_DATA'
 * ibm_db_sa://qcm54853:***@dashdb-txn-sbox-yp-dal09-04.services.dal
Done.
```

Out[12]:

name	coltype	length
Arrest	VARCHAR	5
Beat	SMALLINT	2
Block	VARCHAR	35
Case_Number	VARCHAR	8
Community_Area	DECIMAL	4
Date	VARCHAR	22
Description	VARCHAR	46
District	DECIMAL	4
Domestic	VARCHAR	5
FBI_Code	VARCHAR	3

Summary: -

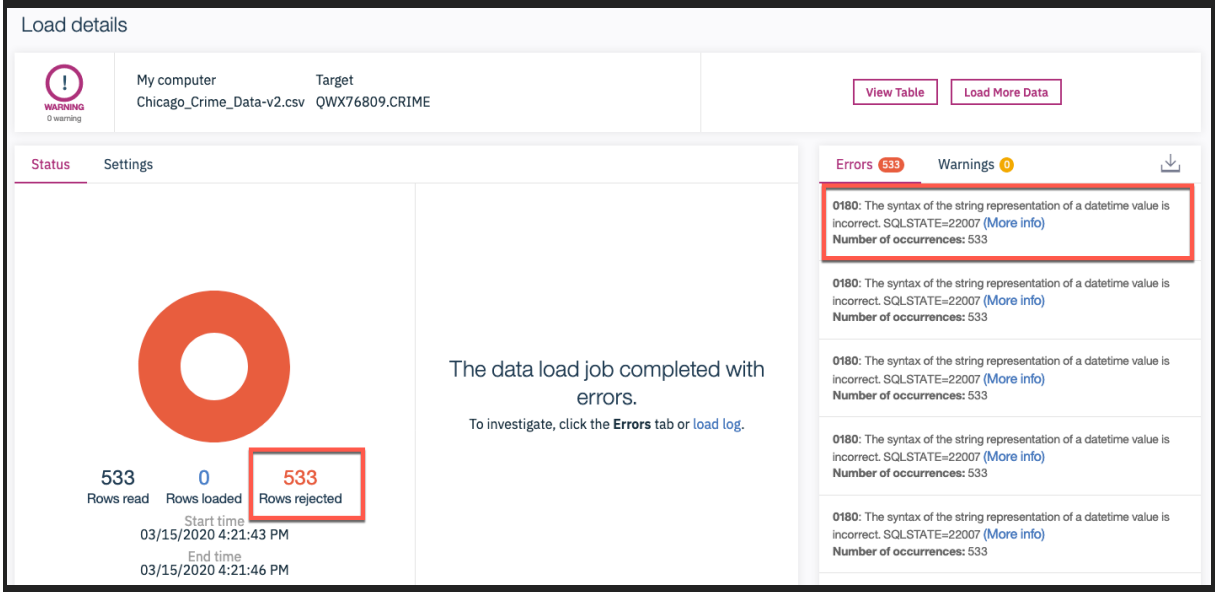
- Metadata or information about database objects such as tables and columns is stored in special tables called system catalog or information schema
- Retrieve a list of tables and their properties by querying "SYSCAT.TABLES" in Db2, In SQL Server it is “INFORMATION_SCHEMA.TABLES” And in Oracle it is “ALL_TABLES” or “USER_TABLES.”
- Retrieve a list of columns and their properties in a Db2 table by querying SYSCAT.COLUMNS or SYSIBM.SYSCOLUMNS.

Loading Data

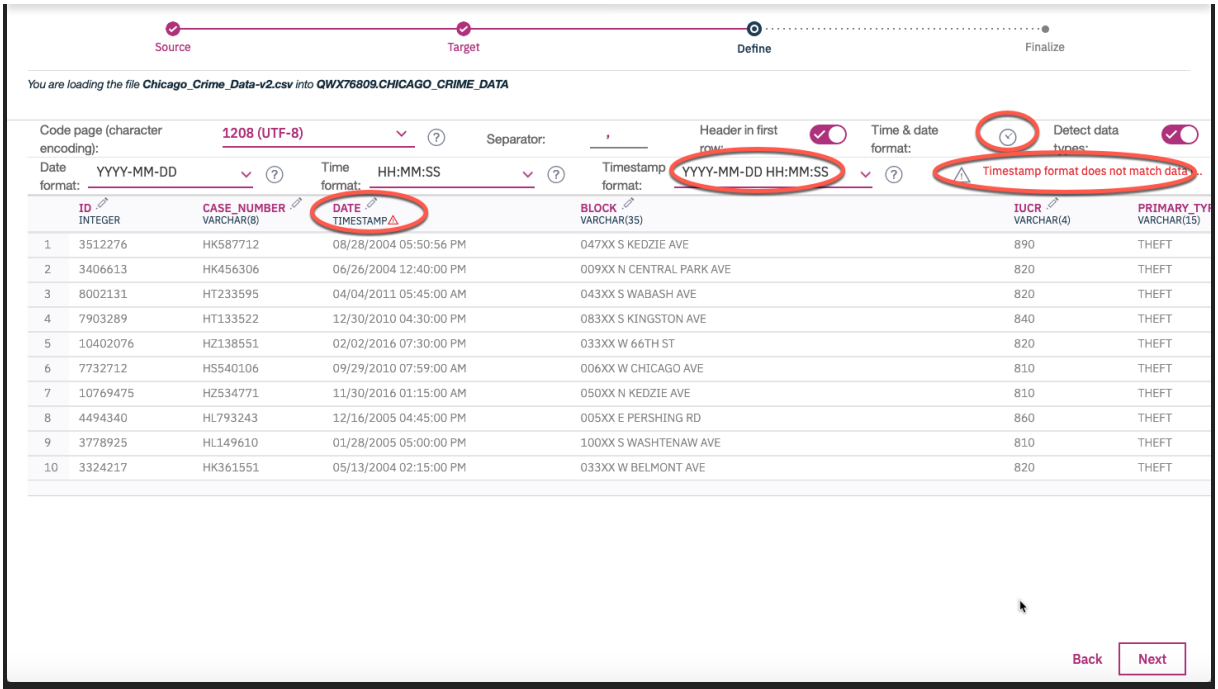
Exercise

When loading data ifrom a CSV file you need to ensure the data in the dataset maps to the correct datatype and format in the database. One area that can be particularly problematic is DATEs, TIMEs, and TIMESTAMPs because their formats can vary significantly.

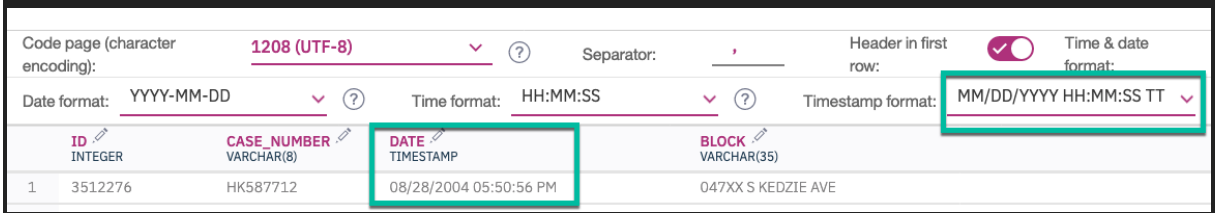
In case the database does not automatically recognize the datatype or format correctly, or the default setting does not match, you will need to manually correct it before loading otherwise you may see an error like the one below when you try to LOAD:



In order to prevent such errors when loading data, in the Db2 console you can preview the datatype and format of the automatically identified values with the values in the datasets in the LOAD screen such as the one below. If there is an issue, it is usually identified with an Warning icon (red triangle with an exclamation mark) next to the datatype of the column (e.g. DATE column in the example below). To correct this, you may first need to click on the "Clock" icon next to the "Time and Date format" to see the formats, if they are not already visible.



First check if there is a pre-defined format in the drop down list that matches the format the date/time/timestamp is in the source dataset. If not, type the correct format. Upon doing so, the Mismatch Warning (and exclamation sign) should disappear. In this example below we changed/overwrote the default Timestamp format of **YYYY-MM-DD HH SS **to **MM/DD/YYYY HH SS TT** to match the value of **08/28/2004 05:50:56 PM** in the dataset.



Week 5

Advanced SQL (Honors)

Module 1: - Views, Stored Procedures, and Transactions (Honors)

1] Views

What is a View?

- Can include specified columns from multiple base tables and existing views.
- Once created, can be queried like a table.
- Only the definition of the view is stored, not the data.

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, ...	100	100000	30001	2
E1002	Alice	James	123457	1972-07-31	F	980 Berry Ln,	200	80000	30002	5
E1003	Steve	Wells	123458	1980-08-10	M	291 Springs,	300	50000	30002	5
E1004	Santosh	Kumar	123459	1985-07-20	M	511 Aurora,	400	60000	30004	5

EMP_ID	F_NAME	L_NAME	ADDRESS	JOB_ID	MANAGER_ID	DEP_ID
E1001	John	Thomas	5631 Rice, ...	100	30001	2
E1002	Alice	James	980 Berry Ln,	200	30002	5
E1003	Steve	Wells	291 Springs,	300	30002	5
E1004	Santosh	Kumar	511 Aurora,	400	30004	5

When should you use a View?

Views can:

- Show a selection of data for a given table.
- Combine two or more tables in meaningful ways.
- Simplify access to data
- Show only portions of data in the table.

Example:
Create a view to display non-sensitive data from the Employees table.

CREATE VIEW Statement –

Syntax with Example:

```
CREATE VIEW <view name> (<column_alias_1>,
<column_alias_2>, … <column_alias_n>)
  AS SELECT <column_1> , <column_2>, … <column_n>
  FROM <table name>
  WHERE <predicate>;
```

```
CREATE VIEW EMPINFO (EMP_ID, FIRSTNAME, LASTNAME, ADDRESS,
JOB_ID, MANAGER_ID, DEP_ID)
  AS SELECT EMP_ID, F_NAME, L_NAME, ADDRESS, JOB_ID,
MANAGER_ID, DEP_ID
  FROM EMPLOYEES;
```

```
SELECT * FROM EMPINFO
```

EMP_ID	FIRST NAME	LASTNAME	ADDRESS	JOB_ID	MANAGER_ID	DEP_ID
E1001	John	Thomas	5631 Rice, …	100	30001	2
E1002	Alice	James	980 Berry Ln,	200	30002	5
E1003	Steve	Wells	291 Springs,	300	30002	5
E1004	Santosh	Kumar	511 Aurora,	400	30004	5

Working with Views –

```
CREATE VIEW EMPINFO (EMP_ID, FIRSTNAME, LASTNAME, ADDRESS,
JOB_ID, MANAGER_ID, DEP_ID)
  AS SELECT EMP_ID, F_NAME, L_NAME, ADDRESS, JOB_ID,
MANAGER_ID, DEP_ID
FROM EMPLOYEES
WHERE MANAGER_ID = '30002'
```

```
SELECT * FROM EMPINFO
```

EMP_ID	FIRST NAME	LAST NAME	ADDRESS	JOB_ID	MANAGER_ID	DEP_ID
E1002	Alice	James	980 Berry Ln,	200	30002	5
E1003	Steve	Wells	291 Springs,	300	30002	5

```
DROP VIEW EMPINFO
```

EMP_ID	FIRST NAME	LAST NAME	ADDRESS	JOB_ID	MANAGER_ID	DEP_ID
E1002	Alice	James	980 Berry Ln,	200	30002	5
E1003	Steve	Wells	291 Springs,	300	30002	5

Summary: -

- Views include specified columns from multiple base tables and existing views
- Views can be queried like a table
- Only the definition of the view is stored, not the data
- The CREATE VIEW statement creates a view based on one or more tables or views.

2] Stored Procedures

What is a Stored Procedure?

- A set of SQL statements stored and executed on the database server
  - Write in many different languages.
  - Accept information in the form of parameters
  - Return results to the client

Benefits of Stored Procedure –

- Reduction in network traffic
- Improvement in performance
- Reuse of code
- Increase in security

CREATE PROCEDURE Statement –

Syntax with Example:

```
CREATE PROCEDURE UPDATE_SAL (IN empNum CHAR(6), IN rating SMALLINT)
  LANGUAGE SQL
  BEGIN
    IF rating = 1 THEN
      UPDATE employees
        SET salary = salary * 1.10
        WHERE emp_id = empNum;
    ELSE
      UPDATE employee
        SET salary = salary * 1.05
        WHERE emp_id = empNum;
    END IF;
  END
```

Working with Stored Procedures –

Call from:

- External applications
- Dynamic SQL statements

```
CALL UPDATE_SAL ('E1001', 1)
```

Summary: -

- Stored procedures are a set of SQL statements that execute on the server
- Stored procedures offer many benefits over sending SQL statements to the server
- You can use stored procedures in dynamic SQL statements and external applications.

3] ACID Transactions

What is a transaction?

- Indivisible unit of work
- Consists of one or more SQL statements
- Either all happens or none

BankAccounts

AccountNumber	AccountName	Balance
B001	Rose	300
B002	James	13450
B003	Shoe Shop	124000
B004	Corner Shop	76000

ShoeShop

Product	Stock	Price
Boots	12	200
High heels	8	600
Brogues	10	150
Trainers	14	300

```
UPDATE BankAccounts
SET Balance = Balance-200
WHERE AccountName = 'Rose';
```

BankAccounts

AccountNumber	AccountName	Balance
B001	Rose	100
B002	James	13450
B003	Shoe Shop	124000
B004	Corner Shop	76000

ShoeShop

Product	Stock	Price
Boots	12	200
High heels	8	600
Brogues	10	150
Trainers	14	300

```
UPDATE BankAccounts
SET Balance = Balance+200
WHERE AccountName = 'Shoe Shop';
```

BankAccounts

AccountNumber	AccountName	Balance
B001	Rose	100
B002	James	13450
B003	Shoe Shop	124200
B004	Corner Shop	76000

ShoeShop

Product	Stock	Price
Boots	12	200
High heels	8	600
Brogues	10	150
Trainers	14	300

```
UPDATE ShoeShop
SET Stock = Stock-1
WHERE Product = 'Boots';
```

- If any of these UPDATE statements fail, the whole transaction must fail.

What is a ACID transaction?

1. **Atomic –** All changes must be performed successfully or not at all.
2. **Consistent –** Data must be in a consistent state before and after the transaction.
3. **Isolated –** No other process can change the data while the transaction is running.
4. **Durable –** The changes made by the transaction must persist.

ACID Commands: -

BEGIN –

- Start the ACID transaction

COMMIT –

- All statements complete successfully.
- Save the new database state.

ROLLBACK –

- One or more statements fail
- Undo changes

```
BEGIN
      UPDATE BankAccounts
      SET Balance = Balance – 200
      WHERE AccountName = 'Rose';

      UPDATE BankAccounts
      SET Balance = Balance + 200
      WHERE AccountName = 'Shoe Shop';

      UPDATE ShoeShop
      SET Stock = Stock - 1
      WHERE Product = 'Boots';
COMMIT
```

```
BEGIN
      UPDATE BankAccounts
      SET Balance = Balance – 200
      WHERE AccountName = 'Rose';

      UPDATE BankAccounts
      SET Balance = Balance + 200
      WHERE AccountName = 'Shoe Shop';

      UPDATE ShoeShop
      SET Stock = Stock - 1
      WHERE Product = 'Boots';
ROLLBACK
```

Calling ACID Commands: -

- Can also be issued by some languages
  - Java, C, R, and Python
  - Requires the use of database specific APIs or connectors
- Use the EXEC SQL command to execute SQL statements from code.

```
void main ()
 {
   EXEC SQL  UPDATE BankAccounts
          SET Balance = Balance – 200
          WHERE AccountName = 'Rose';
   EXEC SQL  UPDATE BankAccounts
          SET Balance = Balance + 200
          WHERE AccountName = 'Shoe Shop';
  EXEC  SQL  UPDATE  ShoeShop
          SET Stock = Stock - 1
          WHERE Product = 'Boots';
  FINISHED:
  EXEC SQL  COMMIT WORK;
  return;

 SQLERR:
  EXEC SQL  WHENEVER SQLERROR CONTINUE;
  EXEC SQL  ROLLBACK WORK;
  return;
}
```

Summary: -

- A transaction represents a complete unit of work, which can be one or more SQL statements.
- In an ACID transaction all the SQL statements must complete successfully or none at all. This ensures the database is always in a consistent state.
- ACID stands for Atomic, Consistent, Isolated, Durable.
- SQL commands BEGIN, COMMIT, and ROLLBACK are used to manage ACID transactions.
- SQL commands can be called from languages like C, R and Python.

Summary & Highlights:-

- Views are a dynamic mechanism for presenting data from one or more tables. A transaction represents a complete unit of work, which can be one or more SQL statements.
- An ACID transaction is one where all the SQL statements must complete successfully, or none at all.
- A stored procedure is a set of SQL statements that are stored and executed on the database server, allowing you to send one statement as an alternative to sending multiple statements.
- You can write stored procedures in many different languages like SQL PL, PL/SQL, Java, and C.

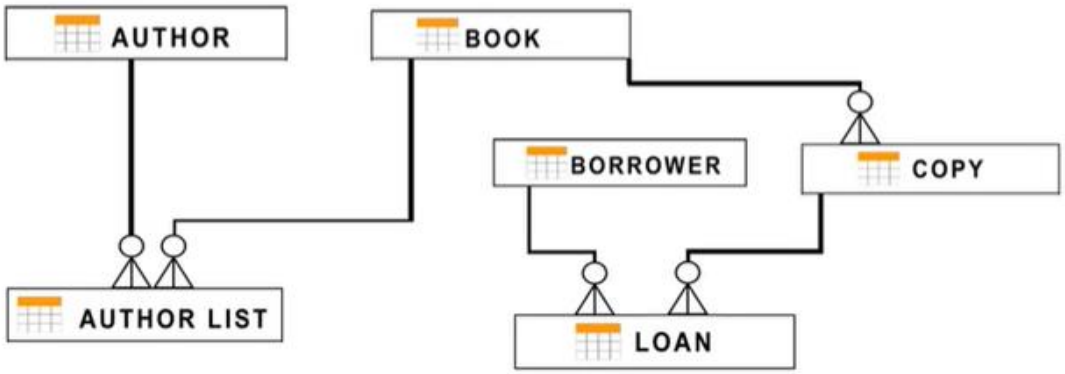
Module 2: - JOIN Statements (Honors)

1] Join Overview

Relational model database diagram: –

JOIN operator –

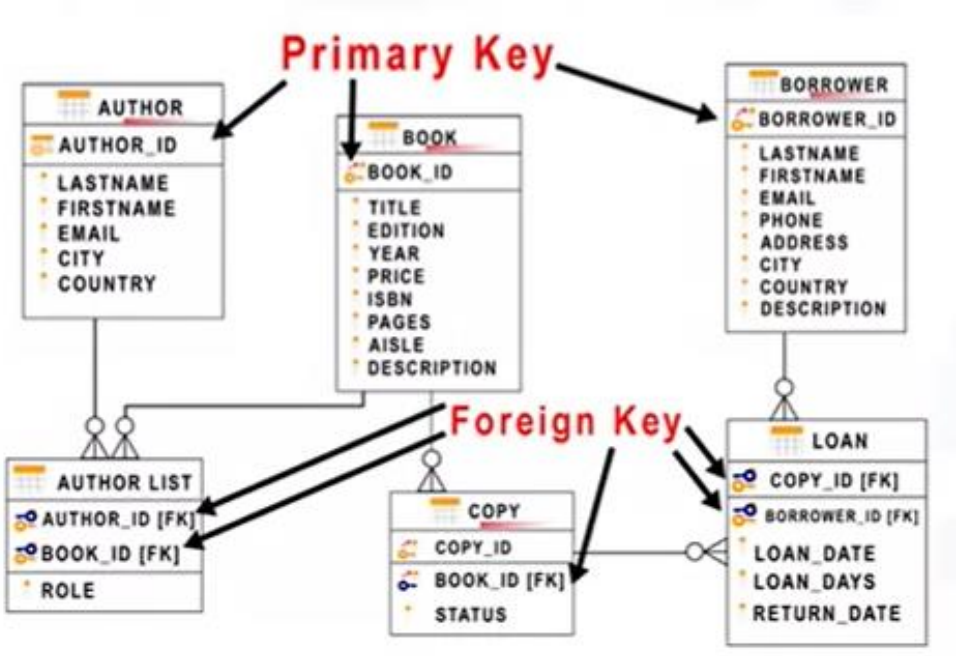
- Combines rows from two or more tables.
- Based on a relationship.



Relational model ER diagram: –

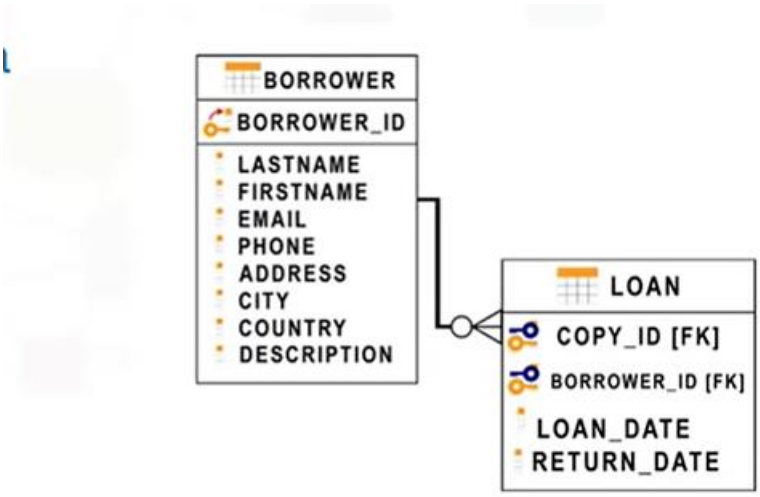
Primary Key – uniquely identifies each row in a table.

Foreign Key – refers to a primary key of another table.



Joining tables –

Which borrower has a book out on loan?

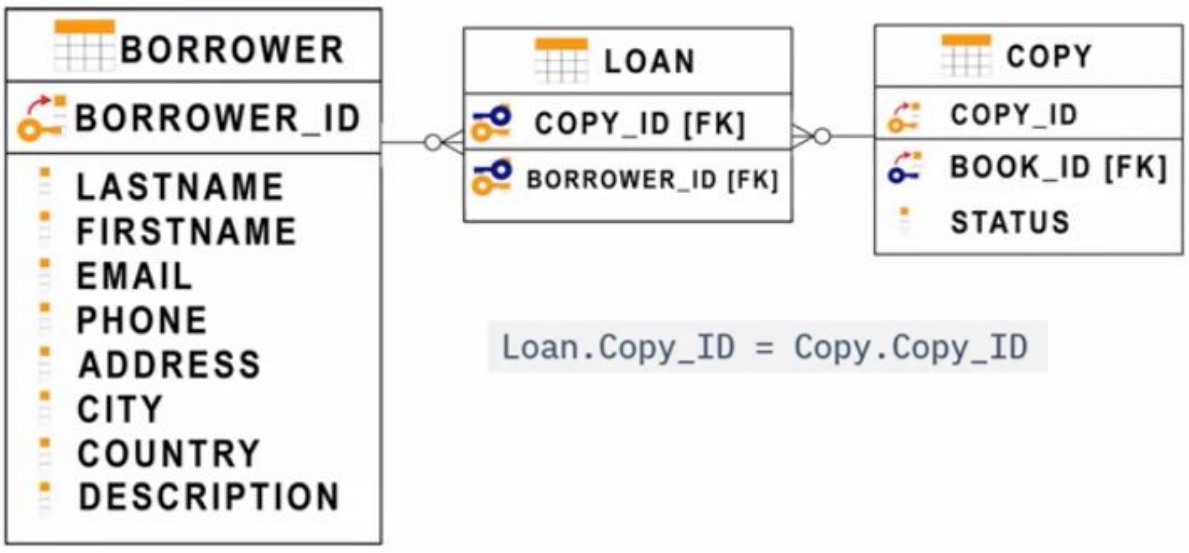



```
Borrower.Borrower_ID = Loan.Borrower_ID
```

Joining Three Tables –

Which copy of a book does the borrower have on loan?

```
Borrower.Borrower_ID = Loan.Borrower_ID
```




```
Loan.Copy_ID = Copy.Copy_ID
```

Types of Joins –

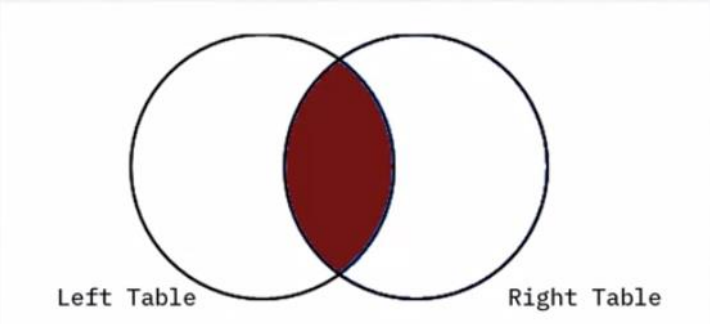
- Inner Join
- Outer Join
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join

Summary: -

- The join operator combines rows from two or more tables.
- The tables being joined are related through Primary and Foreign keys.
- The two types of join are Inner Join and Outer Join.

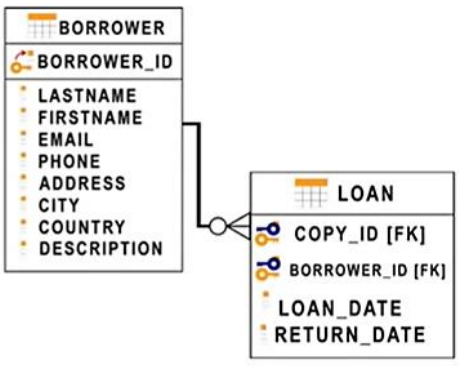
2] *Inner Join*

- Join operations combine the rows from two or more tables.
- Inner join displays match only.



Inner Join Operator –

```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
       L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B INNER JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```



- In this example, the Borrower table is the Left table.
- Each column name is prefixed with an alias to indicate which table each column is associated with.

```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
      L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B INNER JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```

BORROWER_ID	LASTNAME	COUNTRY
D1	SMITH	CA
D2	SANDLER	CA
D3	SOMMERS	CA
D4	ARDEN	CA
D5	XIE	CA
~~D6~~	~~PETERS~~	~~CA~~
~~D7~~	~~LI~~	~~CA~~
~~D8~~	~~WONG~~	~~CA~~
~~D10~~	~~KIEVA~~	~~CA~~

BORROWER_ID	LOAN_DATE
D1	11/24/2010
D2	11/24/2010
D3	11/24/2010
D4	11/24/2010
D5	11/24/2010
~~D9~~	~~11/24/2010~~

```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
      L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B INNER JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```

BORROWER_ID	LASTNAME	COUNTRY	BORROWER_ID	LOAN_DATE
D1	SMITH	CA	D1	11/24/2010
D2	SANDLER	CA	D2	11/24/2010
D3	SOMMERS	CA	D3	11/24/2010
D4	ARDEN	CA	D4	11/24/2010
D5	XIE	CA	D5	11/24/2010

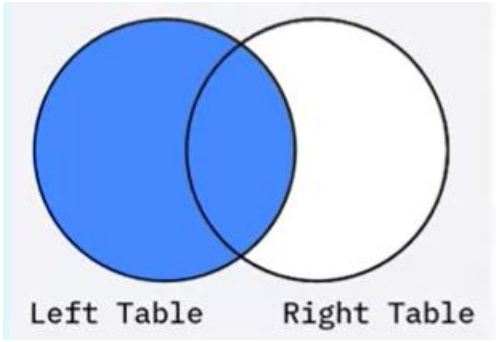
Summary: -

- Inner joins return only rows from joined tables that have a matching value in a common column.
- Rows that do not have a matching value do not appear in the result.

3] Outer Join

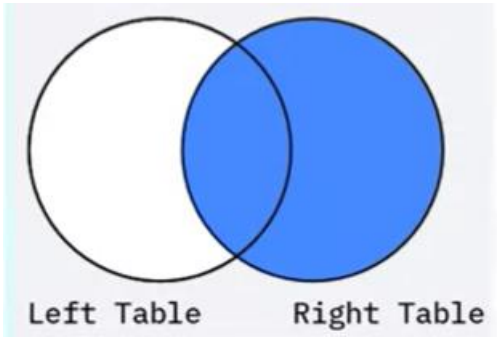
Left Outer Join –

All rows from the left table & any matching rows from the right table



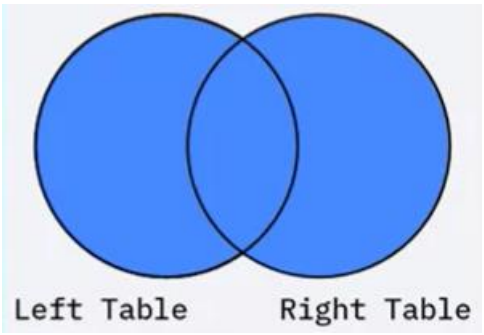
Right Outer Join –

All rows from the right table & any matching rows from the left table.

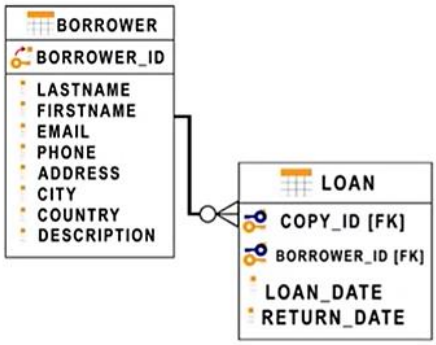


Full Outer Join –

All rows from both tables.

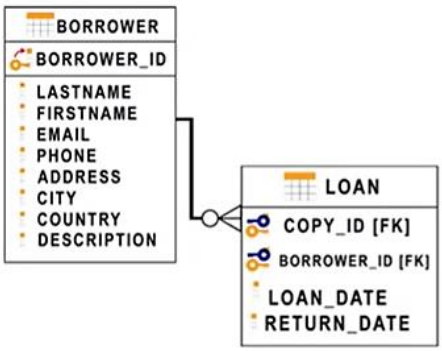


LEFT JOIN Operator –




```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
       L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B LEFT JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```

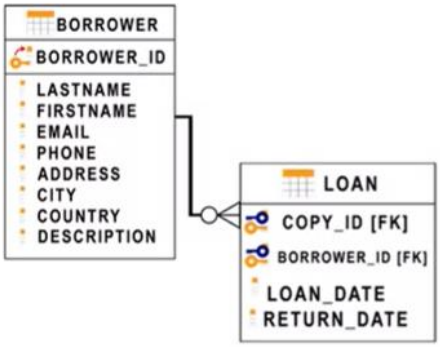
In this example, the Borrower table is the Left table




```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
       L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B LEFT JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```

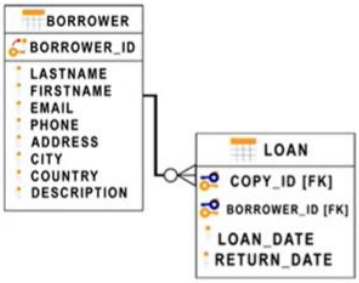
BORROWER_ID	LASTNAME	COUNTRY	BORROWER_ID	LOAN_DATE
D1	SMITH	CA	D1	11/24/2010
D2	SANDLER	CA	D2	11/24/2010
D3	SOMMERS	CA	D3	11/24/2010
D4	ARDEN	CA	D4	11/24/2010
D5	XIE	CA	D5	11/24/2010
D6	PETERS	CA	NULL	NULL
D7	LI	CA	NULL	NULL
D8	WONG	CA	NULL	NULL

RIGHT JOIN Operator –




```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
       L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B RIGHT JOIN LOAN L
     ON B.BORROWER_ID = L.BORROWER_ID
```

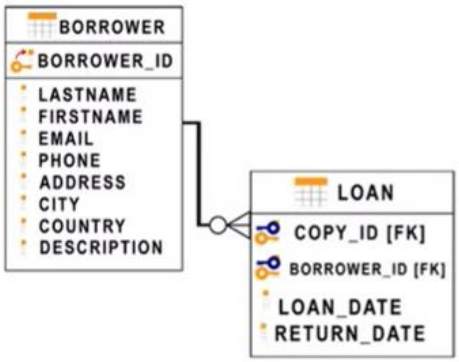
In this example, the Loan table is the right table




```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
       L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B RIGHT JOIN LOAN L
     ON B.BORROWER_ID = L.BORROWER_ID
```

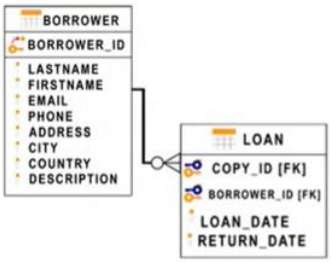
BORROWER_ID	LASTNAME	COUNTRY	BORROWER_ID	LOAN_DATE
D1	SMITH	CA	D1	11/24/2010
D2	SANDLER	CA	D2	11/24/2010
D3	SOMMERS	CA	D3	11/24/2010
D4	ARDEN	CA	D4	11/24/2010
D5	XIE	CA	D5	11/24/2010
NULL	NULL	NULL	D9	11/24/2010

FULL JOIN Operator –




```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
       L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B FULL JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```

In this example, the Borrower table is the Left table




```
SELECT B.BORROWER_ID, B.LASTNAME, B.COUNTRY,
       L.BORROWER_ID, L.LOAN_DATE
FROM BORROWER B FULL JOIN LOAN L
    ON B.BORROWER_ID = L.BORROWER_ID
```

BORROWER_ID	LASTNAME	COUNTRY	BORROWER_ID	LOAN_DATE
D1	SMITH	CA	D1	11/24/2010
D2	SANDLER	CA	D2	11/24/2010
D3	SOMMERS	CA	D3	11/24/2010
D4	ARDEN	CA	D4	11/24/2010
D5	XIE	CA	D5	11/24/2010
D6	PETERS	CA	NULL	NULL
D7	LI	CA	NULL	NULL
D8	WONG	CA	NULL	NULL
NULL	NULL	NULL	D9	11/24/2010

Summary: -

- Left outer joins return all rows from the left table, and all the rows form the right table that match that an inner join would return and all the rows in the first table that do not have a match in the second table.
- Right outer joins return all the rows that an inner join would return and all the rows in the second table that do not have a match in the first table.
- Full outer joins return all matching rows from both tables and all the rows from both tables that don't have a match.

Summary & Highlights:-

- A join combines the rows from two or more tables based on a relationship between certain columns in these tables.
- To combine data from three or more different tables, you simply add new joins to the SQL statement.
- There are two types of table joins: inner join and outer join; and three types of outer joins: left outer join, right outer join, and full outer join.
- The most common type of join is the inner join, which matches the results from two tables and returns only the rows that match.
- You can use an alias as shorthand for a table or column name.
- You can use a self-join to compare rows within the same table.