**INTRODUCTION TO SQL – IBM**

**About**

* 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

What will I learn?

* Basics of SQL and relational databases
* Perform queries using SELECT, INSERT, UPDATE and DELETE statements
* Create a database instance, create tables, and load data
* Filter, order, aggregate your results
* How to write nested queries and join data in many tables
* Analyze real-world data set with SQL
* Confidence to write various types of SQL queries to access and manipulate data.

**Course Overview**

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

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

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

The fundamentals of relational databases and basic SQL commands that you can use to create, manage and query them.

More advanced SQL commands that enable you to group and sort the results of queries, use built-in functions, and include results from multiple tables.

**1.1INTRODUCTION TO DATABASES**

* Basic of SQL (“sikuel”)
* Relational Database Model

Objective:

**SQL:**

SQL is a language used for relational databases to query (query = consultar) or get data out of a database.

**DATA:**

Data is a collection of facts in the form of words, numbers, or even pictures. Data is one of the most critical assets of any business. It is used and collected practically everywhere.

**DATABASE:**

A database is a repository of data. It is a program that stores data. A database also provides the functionality for adding, modifying, and querying that data. There are different kinds of databases of different requirements. The data can be stored in various forms.

**RELATIONAL DATABASE**

|  |  |
| --- | --- |
| **vehículo** | **Color** |
| auto | rojo |

When data is stored in tabular form, the data is organized in tables like in a spreadsheet, which is columns and rows. That's a relational database.

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

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

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

**DBMS (DATA BASE MANAGEMENT SYSTEM)**

RDBMS is a **set of software tools that controls the data such as access, organization, and storage**.

And RDBMS serves as the backbone of applications in many industries including banking, transportation, health, and so on.

Examples of relational database management systems are my SQL, Oracle Database, DB2 Warehouse, and DB2 on Cloud.

**BASIC SQL COMMANDS:** There are five simple commands

1. to **create** a table,
2. **insert** data to populate the table,
3. **select** data from the table,
4. **update** data in the table,
5. **delete** data from the table.

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

**Nota de “Learn SQL”**

A ***relational database*** is a database that organizes information into one or more tables. Here, the relational database contains one table.

A ***table*** is a collection of data organized into rows and columns. Tables are sometimes referred to as *relations*. Here the table is celebs.

A ***column*** is a set of data values of a particular type. Here, id, name, and age are the columns.

A ***row*** is a single record in a table. The first row in the celebs table has:

* An id of 1
* A name of Justin Bieber
* An age of 22

All data stored in a relational database is of a certain data type. Some of the most common data types are:

* INTEGER, a positive or negative whole number
* TEXT, a text string
* DATE, the date formatted as YYYY-MM-DD
* REAL, a decimal value

**Statements**

The code below is a SQL statement. A *statement* is text that the database recognizes as a valid command. Statements always end in a semicolon ;.

**CREATE TABLE table\_name (**

**column\_1 data\_type,**

**column\_2 data\_type,**

**column\_3 data\_type**

**);**

Let’s break down the components of a statement:

1. CREATE TABLE is a *clause*. Clauses perform specific tasks in SQL. By convention, clauses are written in capital letters. Clauses can also be referred to as commands.
2. table\_name refers to the name of the table that the command is applied to.
3. (column\_1 data\_type, column\_2 data\_type, column\_3 data\_type) is a *parameter*. A parameter is a list of columns, data types, or values that are passed to a clause as an argument. Here, the parameter is a list of column names and the associated data type.

The structure of SQL statements vary. The number of lines used does not matter. A statement can be written all on one line, or split up across multiple lines if it makes it easier to read. In this course, you will become familiar with the structure of common statements.

**MANIPULATION**

**Create**

CREATE statements allow us to create a new table in the database. You can use the CREATE statement anytime you want to create a new table from scratch. The statement below creates a new table named celebs.

**CREATE TABLE celebs (  
   id INTEGER,   
   name TEXT,   
   age INTEGER  
);**

1. CREATE TABLE is a clause that tells SQL you want to create a new table.  
2. celebs is the name of the table.  
3. (id INTEGER, name TEXT, age INTEGER) is a list of parameters defining each column, or attribute in the table and its data type:

* id is the first column in the table. It stores values of data type INTEGER
* name is the second column in the table. It stores values of data type TEXT
* age is the third column in the table. It stores values of data type INTEGER

-----

### Column Constraints

Column constraints are the rules applied to the values of individual columns:

* PRIMARY KEY constraint can be used to uniquely identify the row.
* UNIQUE columns have a different value for every row.
* NOT NULL columns must have a value.
* DEFAULT assigns a default value for the column when no value is specified.

There can be only one PRIMARY KEY column per table and multiple UNIQUE columns.

### CREATE TABLE Statement

The CREATE TABLE statement creates a new table in a database. It allows one to specify the name of the table and the name of each column in the table.

### INSERT Statement

The INSERT INTO statement is used to add a new record (row) to a table.

It has two forms as shown:

* Insert into columns in order.
* Insert into columns by name.

### ALTER TABLE Statement

The ALTER TABLE statement is used to modify the columns of an existing table. When combined with the ADD COLUMN clause, it is used to add a new column.

### DELETE Statement

The DELETE statement is used to delete records (rows) in a table. The WHERE clause specifies which record or records that should be deleted. If the WHERE clause is omitted, all records will be deleted.

### UPDATE Statement

The UPDATE statement is used to edit records (rows) in a table. It includes a SET clause that indicates the column to edit and a WHERE clause for specifying the record(s).

**1.2 SELECT Statement - Retrieving Data from a Table**

**RETRIEVING ROWS OF TABLE**

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

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

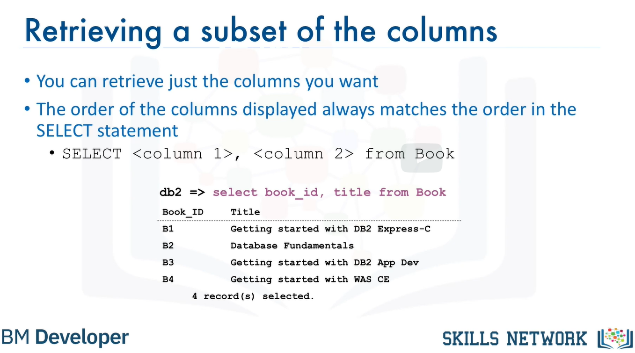
To see the data, we use the SELECT statement.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamenteThe **SELECT** statement is a data manipulation language statement.

**Data Manipulation Language statements or DML** statements are used to read and modify data.

**The SELECT statement is called a query**, and **the output** we get from executing this query is called a result set or **a result table**. In its simplest form, a SELECT statement is select star from table name.

**db2=>select\*from B00k** 🡪 selecciona todas las columnas

**db2=>select <column N> from B00k 🡪** se elige la columna deseada.

**select <column X>, <column y> from B00k** 🡪Retrieve a subset of columns.

You can retrieve just a subset of columns. If you want, you can retrieve just two columns from the table book.

For example book\_id and title. In this case, the select statement is select book\_id, title from book. In this case, only the two columns display for each of the four rows. Also notice that the order of the columns displayed always matches the order in the SELECT statement.

**Restricting the result set: WHERE clause**

However, what if we want to know the title of the book whose book\_id is B1. Relational operation helps us in restricting the result set by allowing us to use the clause WHERE.

The WHERE clause always requires a predicate. A predicate is conditioned evaluates to true, false or unknown. Predicates are used in the search condition of the WHERE clause.

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

**WHERE** B00k\_ID = “B1”

There are several **comparison operators** supported bya relational database management system:

|  |  |
| --- | --- |
| equal to | = |
| greater than | > |
| less than | < |
| greater than or equal to | >= |
| less than or equal to | <= |
| not equal to | <> |

**1.3 LAB**

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'; \*\*

**Cuando trabajas en SQL, recuerda que al terminar un ”Statement”, debes escribir “;” (semicolon) para que se ejecute el comando siguiente.**

**En este LAB usaremos “[Datasette](https://github.com/simonw/datasette?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDB0201ENSkillsNetwork20127838-2022-01-01)”.** **An open source multi-tool for exploring and publishing data.**

**Trabajamos con** [**Film\_locations\_SF**](https://data.sfgov.org/Culture-and-Recreation/Film-Locations-in-San-Francisco/yitu-d5am?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDB0201ENSkillsNetwork20127838-2022-01-01)**.**

**1.4 COUNT, DISTINCT, LIMIT**

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

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

Let's say you create a table called MEDALS which has a column called COUNTRY, and you want to retrieve the number of rows where the medal recipient is from Canada.

You can issue a query like this:

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

**DISTINCT:** DISTINCT is used to remove duplicate values from a result set. Example, to retrieve unique values in a column, select DISTINCT columnname from tablename.

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

Example, retrieve the list of unique countries that received gold medals. That is, removing all duplicate values of the same country.

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

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

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

**Select \* from tablename LIMIT 10.**

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

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

**Select \* from MEDALS where YEAR = 2018 LIMIT 5.**

**1.5. LAB #2**

**1.6. INSERT STATEMENT**

* ADDS INFO TO THE TABLE:. Helps to populate the database table.
* It is a form of Data Manipulation Language Statement (DML Statements)
* Tables do not need to be populated one row at a time. Multiple rows can be inserted by specifying each row in the values clause.

Ejemplo:mario.aquiles89

|  |  |
| --- | --- |
| formula | ejemplo |
| 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”)  (“A2”, “Ahuja”, “Rav”, [ra@ibm.com](mailto:ra@ibm.com), “Toronto”, “CA”) |

**UPDATE STATEMENT**

* Altering data in a relational database in a DML statement

|  |  |
| --- | --- |
| Formula | Ejemplo |
| UPDATE [Tablename]  SET [[ColumnNamee]=[Value]]  <WHERE [Condition]> | UPDATE Author  SET LastName=”Katta”  Firstname=”Lakshmi”  Where Author\_ID=”A2” |

**DELETE STATEMENT**

* Deletes data in a relational database in a DML statement

|  |  |
| --- | --- |
| Formula | Ejemplo |
| DELETE FROM [TableName]  <WHERE [Condition]> | DELETE FROM Author  Where Author\_ID IN (“A2”, ”A3”) |

**RELATIONAL DATABASE CONCEPTS**

Objectives:

* Advantages of relational model.

Allows data independence. Data is stored in tables which provides logical data independence, physical data independence and physical storage independence.

* How the entity name and attributes map to a relational database table

Diagrama

Descripción generada automáticamenteUsing an **ENTITY RELATIONSHIP** data model (or ER data model) is an alternative to a relational data model. Using a simplified library database as an example this figure shows **an ERD** (Entity Relationship Diagram) that represents entities called tables and their relatinoships.

In the library example we have books, which may have many authors and the library may have several copies. But each copy may be borrowed by one borrower at a time.

An ER model proposes thinking of a database **as a collection of entities**. Rather than being used as a model, it’s a tool to design relational databases. *In the ER model entities are objects that exists independently of any other entities in the database*.

* Describe the difference between an entity and an attribute

**Diagrama

Descripción generada automáticamenteThe building blocks of an ER diagram are entities and attributes.** An entity can be a noun: person, place or thing. In a ER diagram, an entity is drawn as a rectangle.

Entities have **attributes** that are data elements that characterize the entity. Attributes tell us more about the entity, and are drawn as ovals.

**Entities map to tables in a relational database.**

**Attributes map to columns in a table.**

* Identify commonly used data types

In this case, book is an example of an entity, attirubtes are properties or characteristics of an entity, such as title, edition, etc.

**Common data types include characters, numbers, and dates/times**

Attributes are connected to exactly one entity. **The entity books becomes a table in the data base and the attributes become the columns in a table**.

A table is a combination of rows and columns. Attributes get translated into columns in a table providing the actual table form of rows and columns.

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

* Describe the function of primary keys

**Primary** **keys**: each table is assigned a primary key; **the primary key of a relational table uniquely identifies each tuple or row in a table, preventing duplication of data** and providing a wat of defining relationships between tables.

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

**How to create a Database instance on Cloud**

in order to learn SQL you first need to have a database available to practice your SQL Queries, an easy way to do so is to create an instance of a database ina cloud and use it to execute your SQL queries.

* CLOUD DATABASE BASICS

A cloud database is a database service built and accessed through a cloud platform.

Advantages:

**ease of use**, user can access could database from anywhere using a vendor´s **API** or **web** **interface**, or your **own applications whether on cloud or remote**.

**Scalability&Economics:** cloud datbases can expand an shrink storage and capacity to accommodate changing needs and demands, so you pay what you actually use.

**Disaster recovery:** cloud backups and geographical distribution.

* LIST SOME CLOUD DATABASES
  + IBM Db2
  + Databases for PostgreSQL
  + Oracle DatabaseCloud Service
  + Microsoft azure SQL Database
  + Amazon relational Database Services (RDS)

These databases can run in the cloud **as a Virtual Machine (VM) which you can manage or delivered as a Managed service**, depending on the vendor.

The database services can either be **single or multi-tenant depending on the service plan**.

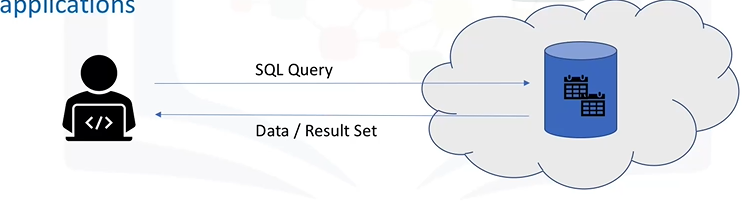
* DESCRIBE A DATABASE INSTANCE

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

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

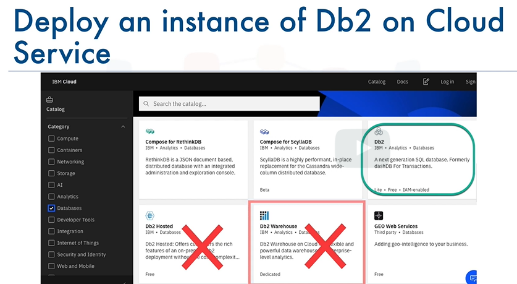
*The database service instance will hold your data* ***in related tables.***

*Once your data is loaded into the database instance, you can connect to the database instance using a web interface or APIs in your applications*. *Any data retrieved is returned to the application as a result set.*



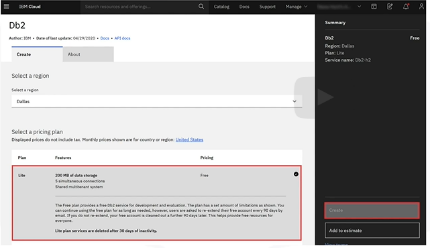
* CREATE AN INSTANCE OF IBM Db2 on Cloud

**IBM Db2** on Cloud is a SQL database provisioned for you in the Cloud.

**Deploy an instance od Db2 on Cloud Service:**

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**

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:** You can view the IBM Db2 service that you created by selecting services from your IBM Cloud dashboard. From this dashboard, you can manage your database instance.

The Web Console allows you to create tables, load data, explore data in your tables, and issue SQL queries.

In order to access your database instance from your applications, you will need the service credentials.

For the first time around, you'll need to create a set of new credentials. You can also choose to create multiple sets of credentials for different applications and users. Once a set of service credentials is created, you can view it as adjacent snippet.

The credentials include the necessary details to establish a connection to the database, and includes the following; a database name and port number, a host name, which is the name of the server on the Cloud on which your database instance resides, a username, which is the user ID you'll use to connect along with the password.

**TYPES OF SQL STATEMENTS. DDL VS. DML**

* DATA DEFINITION LANGUAGE VS. DATA MANIPULATION LANGUAGE STAMENTS.

SQL Statements are used for interacting with Entities (that is, tables), Attributes (that is, columns) and their tuples (or rows with data values) in relational databases.

SQL statements fall into two different categories:

Data Definition Language statements and Data Manipulation Language statements.

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

**COMMON DDL STATEMENT TYPES INCLUDE CREATE, ALTER, TRUNCATE, AND DROP.**

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

|  |  |
| --- | --- |
| Formula | Ejemplo |
| CREATE TABLE Table\_name  (column\_name datatype optional\_parameters, column\_name\_2 datatype,  …  Column\_name\_n datatype  ) | CREATE TABLE *provinces (*  Id char{character} (2) PRIMARY KEY NOT NULL,  Name varchar {*variable character*} (24)  *)*  *Ej2.*  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)  ) |

asc

***\*\*\* “NOT NULL”****constraint added after the datatype - means that it cannot contain a NULL or an empty value.*

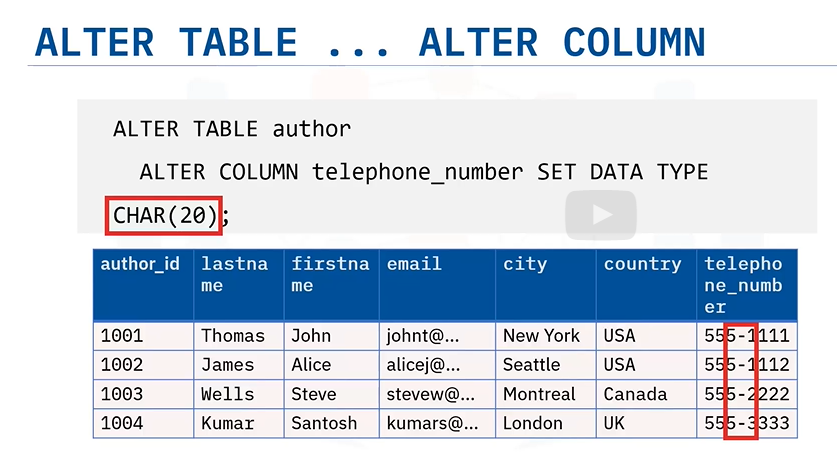
**ALTER**: is used for altering tables including adding and dropping columns and modifying their datatypes;

|  |  |
| --- | --- |
| Formula | Ejemplo |
| ALTER TABLE <table\_name>  ADD COLUMN <column\_name\_1> datatype  …..  ADD COLUMN <column\_name\_n> datatype | ALTER TABLE author  ADD COLUMN telephone\_number \*BIGINT  Ej.2  ALTER TABLE <table\_name>  ALTER COLUMN <column\_name> SET DATA TYPE <datatype>; |

Differently to “CREATE” statement though, you do not use parentheses to enclose the parameters for the ALTER TABLE statement. <table\_name>

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

**\*\*\*BIGINT: can hold a number up to 19 digits long.**

**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.**

**DROP**: is used for deleting tables.

|  |  |
| --- | --- |
| Formula | Ejemplo |
| ALTER TABLE <table\_name>  DROP COLUMN <column\_name\_1> | Alter table AUTHOR  DROP COLUMN telephone\_number; |

If your spec changes and you no longer need this extra column, you can again use the ALTER TABLE statement, this time with the DROP COLUMN clause, to remove the column as shown:

ALTER TABLE author DROP COLUMN telephone\_number; Similar to using DROP COLUMN to delete a column from a table, you use the DROP TABLE statement to delete a table from a database.

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

**DROP TABLE:** Similar to using DROP COLUMN to delete a column from a table, you use the DROP TABLE statement to delete a table from a database. If you delete a table that contains data, by default the data will be deleted alongside the table.

|  |  |
| --- | --- |
| Formula | Ejemplo |
| DROP TABLE <table\_name>; | DROP TABLE author; |

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

|  |  |
| --- | --- |
| Formula | Ejemplo |
| TRUNCATE TABLE <table\_name>  \*INMEDIATE; |  |

***\*\*\*The IMMEDIATE specifies to process the statement immediately and that it cannot be undone***.

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

*These are also sometimes referred to as CRUD operations,* that is, **C**reate, **R**ead, **U**pdate and **D**elete rows in a table.

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

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

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

**UPDATE**: edits row or rows in a table;

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

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

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

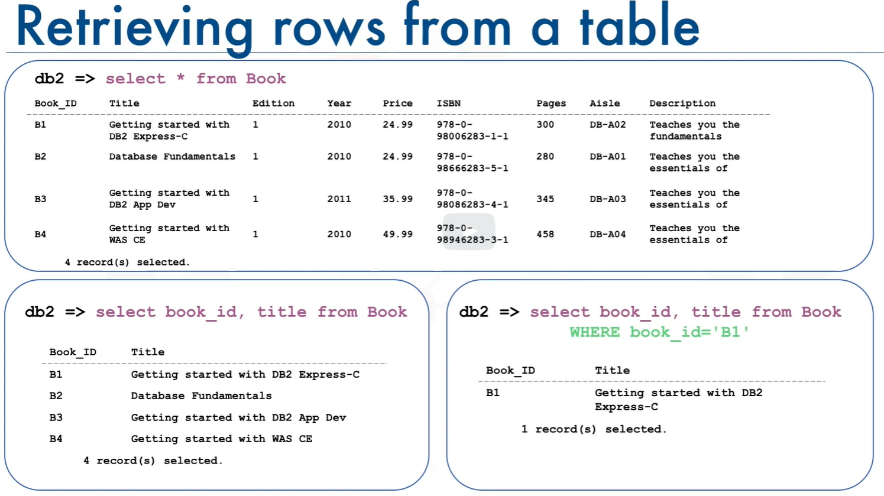
**MODULE 3 – INTERMEDIATE SQL**

**USING STRING PATTERNS, RANGES**

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

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

In its simplest form, a SELECT statement is select star from table name.

Based on a simplified library database model and the table Book, SELECT star from Book gives a result set of four rows.

All the data rows for all columns in the table Book are displayed or you can retrieve a subset of columns for example, just two columns from the table book such as Book\_ID and Title.

Or you can restrict the result set by using the WHERE clause. For example, you can select the title of the book whose Book\_ID is B1.

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

The WHERE clause always requires a predicate, which is a condition that evaluates to true, false, or unknown**. But what if we don't know exactly what value the predicate is?**

For example, what if we can't remember the name of the author, but we remember that their first name starts with R?

**Answer:**

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**.

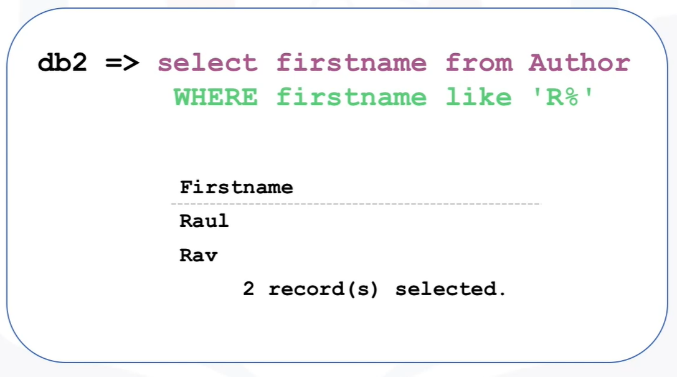
If we can't remember the name of the author, but we remember that their name starts with R, we use the WHERE clause with the like predicate.

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.

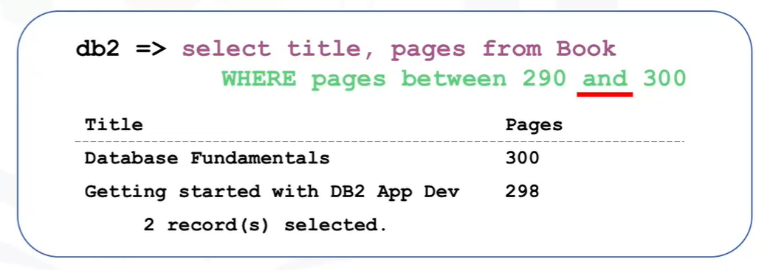
The percent sign can be placed before the pattern, after the pattern, or both before and after the pattern.

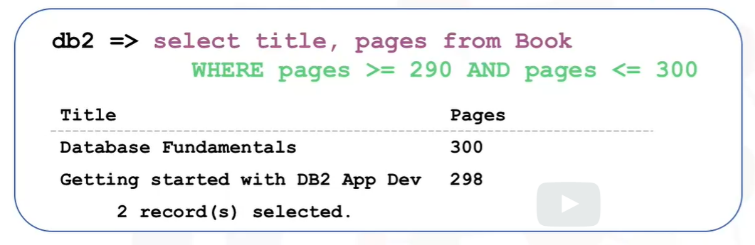
|  |  |
| --- | --- |
| FORMULA | EJEMPLO |
| WHERE firstname LIKE R% | **WHERE** firstname LIKE **R%** |

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.

**AND WIDE PARAMETERS OF SEARCH?**

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

****We could write the SELECT statement like this, specifying the WHERE clause as, where pages is greater than or equal to 290, and pages is less than or equal to 300.

But in a relational database, we can use a range of numbers to specify the same condition.

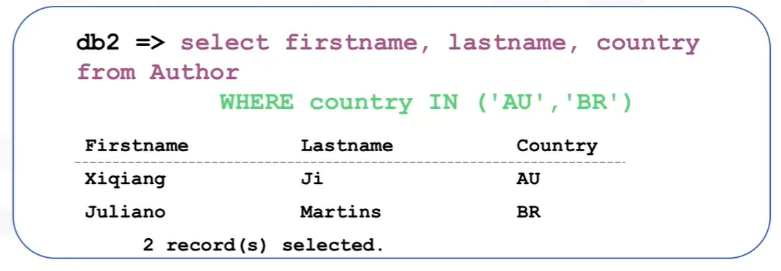
Instead of using the comparison operators greater than or equal to, we use the comparison operator **'BETWEEN AND.'**

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

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

For instance, when listing countries, the WHERE clause would become very long repeatedly listing the required country conditions.

**IN OPERATOR**

****Instead, we can use the IN operator. **The IN operator allows us to specify a set of values in a WHERE clause.** This operator takes a list of expressions to compare against. In this case the countries Australia or Brazil.

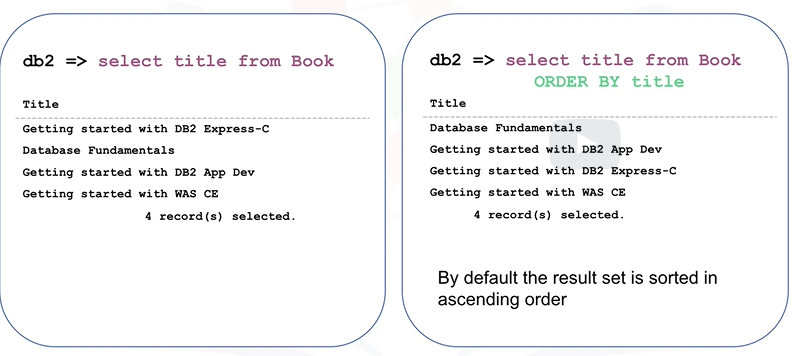
**SORTING RESULT SETS: Para ordenar en orden alfabético los resultados**

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

In its simplest form, a select statement is select \* from table name.

Based on our simplified library database model, in the table book, select \* from book gives a result set of four rows.

All the data rows for all columns in the table book are displayed. We can choose to list the book titles only as shown in this example, select title from book.

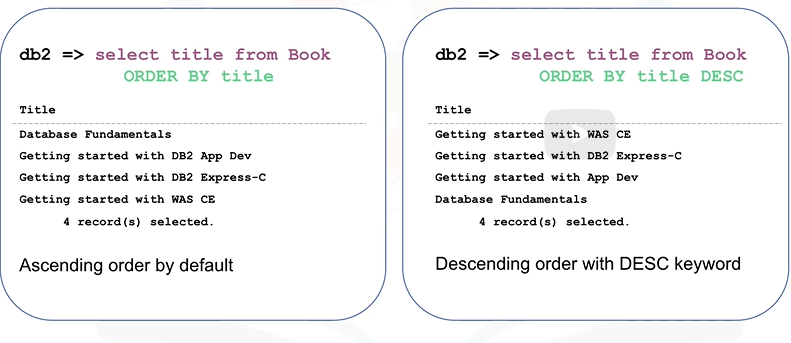
However, the order does not seem to be in any order. Displaying the results set in alphabetical order would make the result set more convenient.

To do this, we use **the "ORDER BY" clause.** To display the result set in alphabetical order, we add the order by clause to the select statement. The order by clause is used in a query to sort the result set by a specified column.

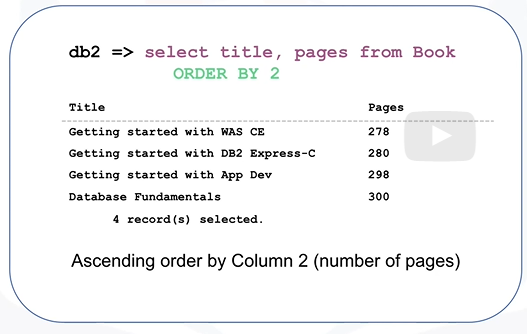
In this example, we have used order by on the column title to sort the result set. By default, the result set is sorted in ascending order. In this example, the result set is sorted in alphabetical order by book title.

To sort in descending order, **use the key word “DESC"** The result set is now sorted according to the column specified, which is title, and is sorted in descending order.

Notice the order of the first three rows.



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

****In this example, select title, pages from book **“ORDER BY 2”**, indicates the column sequence number in the query for the sorting order.

Instead of specifying the column name pages, the number two is used.

In the select statement, the second column specified in the column list is pages, so the sort order is based on the values in the pages column. In this case, the pages column indicates the number of pages in the book.

As you can see, the result set is in ascending order by number of pages. Now you can describe how to sort the result set by either ascending or descending order and explain how to indicate which column to use for the sorting order.

**GROUPING RESULT SETS**

We will learn techniques in retrieving data from a relational database table, and sorting and grouping how the results set displays. How to eliminate duplicates from a result set, describe how to further restrict a result set.

**DISTINCT CLAUSE**

Permite eliminar los duplicados.

|  |  |
| --- | --- |
| Formula | Ejemplo |
| db2=> SELECT DISTINCT (column) FROM dataset | db2 => SELECT DISTINCT (country) FROM Author |

**GROUP CLAUSE**

Si hay más de un autor que viene de un país, podemos agruparlos por país.

En otras palabras, group clause permite saber cuántos elementos de cada categoría en la columna existen.

|  |  |
| --- | --- |
| Formula | Ejemplo |
| Db2=> SELECT element1, COUNT (element1) FROM dataset **GROUP BY** element1 | Db2=>SELECT country, COUNT (country) FROM author **GROUP BY** country |

**AS Clause**

Serves to assign a column name to the result set.

|  |
| --- |
| Formula |
| Db2=> SELECT country, COUNT (country) **AS COUNT** FROM author GROUP BY country |

**HAVING Clause**

When we have the number of authors by countries, we can further restrict the number of rows by passing some conditions.

To set a condition to a “group by” clause we use the keyword “HAVING”

|  |
| --- |
| Fórmula |
| Db2=>SELECT country, COUNT (country) AS COUNT FROM author  **GROUP BY country HAVING COUNT (country) >4** |

In this example we want to check countries with 4 or more authors: “*Having count country greater than four*”

**BUILT IN DATABASE FUNCTIONS**

They are faster and reduce the amount of data needed.

**AGGREGATE OR COLUMN FUNCTIONS**

These functions collect info in an INPUT amd then produce an output with a sample:

INPUT : Collection of Values

OUTPUT : Single Value

Examples:

SUM(), MIN(), MAX(), AVG(), etc

**SUM()**

Add up all the values in a column.

|  |  |
| --- | --- |
| Formula | Example |
| SUM(COLUMN\_NAME)  SELECT SUM(COLUMN\_NAME) FROM \_DATASET\_ | Select SUM(COST) FROM PETRESCUE  Result:  1  1718.24 |

**COLUMN ALIAS**

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

However, it is possible to explicitly name the result column. For instance, If we want to name the column result “Sum\_of\_cost”, instead of “1”, we will write:

|  |
| --- |
| Formula |
| select SUM(COST) as SUM\_OF\_COST from PETRESCUE. |

**MIN AND MAX.**

Minimum, as the name implies, is used to get the lowest value. Similarly, Maximum is used to get the highest value. For example, to get the maximum quantity of any animal rescue in a single transaction:

|  |
| --- |
| Formula |
| select MAX(QUANTITY) from PETRESCUE. |

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

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

|  |
| --- |
| Formula |
| select MIN(ID) from PETRESCUE where animal = “dog”. |

**AVERAGE – AVG()**

The Average (AVG) function is used to return the average or the mean value.

For example, to specify the average value of cost, as:

|  |
| --- |
| Formula |
| select AVG(COST) from PETRESCUE. |

**SCALAR AND STRING FUNCTIONS**

Scalar functions perform operations on individual values.

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

Example: Round up or Down every value in COST:

|  |
| --- |
| Formula |
| SELECT ROUND(COST) FROM PETRESCUE |

There is a class of scalar functions called string functions, that can be used for operations on strings. That is char and varchar values.

For example, to retrieve the length of each value in animal column,

|  |
| --- |
| Formula |
| SELECT LENGTH(ANIMAL) FROM PETRESCUE |

**UCASE, LCASE**

Uppercase and lowercase functions can be used toreturn uppercase or lowercase values of strings.For example, to retrieve animal values in uppercase:

|  |
| --- |
| Formula |
| select UPPERCASE (ANIMAL) from PETRESCUE. |

Scalar functions can be used in the where clause. For example, to get lowercase values of the animal column for cat:

|  |
| --- |
| Formula |
| select \* from PETRESCUE where LOWERCASE(ANIMAL) = “cat” |

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

You can also have one function operate on the output of another function. For example, to get unique cases for the animal column in uppercase:

|  |
| --- |
| Formula |
| select DISTINCT (UCASE(ANIMAL)) from PETRESCUE. |

**DATE AND TIME BUILT-IN ACTIONS:**

Most databases contain special data types for dates and times. In db2 we have the following:

|  |  |
| --- | --- |
| DATE | **YYYYMMDD** |
| TIME | **HHMMSS** |
| TIMESTAMP | **YYYYXXDDHHMMSSZZZZZZ** |
| DATE/TIME FUNCTIONS | **YEAR(), MONTH(), DAY(), DAYOFMONTH(), DAYOFWEEK(),DAYOFYEAR(),WEEK(),HOUR(), MINUTE(), SECOND()** |

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

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

|  |
| --- |
| Formula |
| select DAY (RESCUEDATE) from PETRESCUE where ANIMAL = cat. |

Date and time functions can be used in the where clause. For example, to get the number of rescues during the month of May, that is, from Month 5,

|  |
| --- |
| Formula |
| select COUNT \* from PETRESCUE where MONTH (RESCUEDATE) = “05”. |

You can also **perform date or time arithmetic**. For example, to find out what date it is three days after each rescue date, maybe you want to know this because the rescue needs to be processed within three days.

|  |
| --- |
| Formula |
| select (RESCUEDATE + 3 DAYS) from PETRESCUE. |

**Special registers** current time and current date are also available.

Special register: CURRENT\_DATE, CURRENT\_TIME

For example, to find how many days have passed since each rescue date till now

Example:

|  |
| --- |
| Formula |
| Select (CURRENT\_DATE - RESCUEDATE) from PETRESCUE.  Result (Format YMMDD)  10921 |

The result will be in years, months, days.

**SUBQUERIES and NESTED SELECTS**

Sub-queries or sub selects are like regular queries but placed within parentheses and nested inside another query. This allows you to form more powerful queries than would have been otherwise possible.

|  |
| --- |
| Ejemplo |
| SELECT column1 FROM table  Where COLUMN = (SELECT max (column2) from TABLE) |

More examples: Let's consider a scenario which may necessitate the use of sub-queries. Let's say, we want to retrieve the list of employees who earn more than the average salary. To do so, you could try this code.

|  |
| --- |
| Example |
| SELECT\* from Employees  where salary > AVG (salary) |
| RESULTS: This query will result in error: SQL0120N Invalid use of an aggregate function or OLAP function. SQLCODE=-120, SQLSTATE=42903 |

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.

|  |
| --- |
| Example |
| Select EMP\_ID, F\_NAME, L\_NAME, SALARY  from employees  where SALARY <  (select AVG(SALARY) from employees); |

Notice that the average function is evaluated in the first part of the sub-query. Allowing us to circumvent the limitation of evaluating it directly in the WHERE clause. The sub-select doesn't just have to go in the WHERE clause.

**SUB QUERIES IN LIST OF COLUMNS**:

* Substitute column name with a subquery
* Called column expressions

Say we wanted to compare the salary of each employee with the average salary.

|  |
| --- |
| Example |
| Select EMP\_ID, SALARY, AVG (SALARY) AS AVG\_SALARY  from employees ; |
| RESULT: Error: no group by clause is specified |

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

|  |
| --- |
| Example |
| Select EMP\_ID, SALARY,  (select AVG(SALARY) from employees )  AS AVG\_SALARY  From employees ; |

**Subqueries in FROM Clause**

Another option is to make the sub-query be part of the FROM clause. Sub-queries like these are sometimes called derived tables or table expressions. Because the outer query uses the results of the sub-query as a data source.

* Substitute the table name with a sub-query
* Called **derived tables or table expressions: because the outter querry uses the result of the subqyery as a data source**

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

|  |
| --- |
| Example |
| Select \* from  (select EMP\_ID, F\_NAME, L\_NAME, DEP\_ID  From employees) AS EMP4ALL ; |

In this lesson we learnt:

* How subqueries and nested queries form richer queries.}
* How they can overcome some o limitations of aggregate functions
* How to use subqueries in the
  + WHERE clause
  + List of columns
  + FROM clause

HOW DOES A TYPICAL NESTED SELECT STATEMENT SYNTAX LOOKS LIKE:

SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

WHERE column\_name OPERATOR

(SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

WHERE condition);

**WORKING WITH MULTIPLES TABLES**

Ways to access multiple tables in the samequery:

1. Sub queries
2. Implicit JOIN
3. JOIN operators (Inner Join, Outer Join, etc.)

**Accessing Multiple Tables with subqueries**

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

|  |
| --- |
| Formula |
| Select \* from employees  Where DEP\_ID IN  ( select DEPT\_ID\_DEP from departments ); |

To retrieve only the list of employees from a specific location where Employees table does not contain location information, you need to get the location info from another table: Departments table.

|  |
| --- |
| Example |
| Select \* from employees  Where DEP\_ID IN  ( select DEP\_ID\_DEP from departments  Where LOC\_ID = “L0002” ) ; |

**Multiple tables with subqueries**

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

|  |
| --- |
| Example |
| Select DEPT\_ID\_DEP, DEP\_NAME from departments  Where DEPT\_ID\_DEP IN  (select DEP\_ID from employees  Where SALARY > 70000 ) ; |

**Accessing multiple tables with Implicit Join**

Specify 2 tables in the FROM clause:

|  |
| --- |
| Example |
| Select \* from employees, departments; |

The result is a full join (or cartesian join) because every row in the first table is joined with every row in the second table.

Also, the result set will have more rows than in both tables.

Also, it is possible to use additional operand to limit the result set: *for instance, we limit the result set to only rows matching with the departments iD*

|  |
| --- |
| Example |
| Select \* from employees, departments  Where **employees.**DEP\_ID\_DEP =  **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 is possible that different tables could have the column names that are exactly the name.

Since some table names can be long, we can use shorter aliases for table names like shown here:

Shorter Aliases for the table names:

|  |
| --- |
| Example |
| Select \* from employees E, departments D  Where E.DEP\_ID = D.DEPT\_ID\_DEP; |

Alias E: employees table

Alias D: Department tables

**Accessing multiple tables with implicit Join**

To see the department name for each employee:

|  |
| --- |
| Example |
| Select EMP\_ID, DEP\_NAME  From employees E, departments D  Where E.DEP\_ID = D.DEPT\_ID\_DEP; |

Also, similar to before the column names can be prefix with

|  |
| --- |
| Example |
| Select E.EMP\_ID, E.DEP\_NAME  From employees E, departments D  Where E.DEP\_ID = D.DEPT\_ID\_DEP; |

RECAP - Hands-on Lab: Working with Multiple Tables (30m)

**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;

LAB:

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

**Select** \* **from** employees **where** JOB\_ID **IN** (**select** JOB\_IDENT **from** jobs);

* Retrieve only the list of employees whose JOB\_TITLE is Jr. Designer.

**select** \* **from** employees **where** JOB\_ID **IN** (**select** JOB\_IDENT **from** jobs **where** JOB\_TITLE= 'Jr. Designer');

* Retrieve JOB information and who earn more than $70,000.

**select** JOB\_TITLE, MIN\_SALARY,MAX\_SALARY,JOB\_IDENT **from** jobs **where** JOB\_IDENT **IN** (**select** JOB\_ID **from** employees **where** SALARY > 70000 );

* Retrieve JOB information and whose birth year is after 1976.

**select** JOB\_TITLE, MIN\_SALARY,MAX\_SALARY,JOB\_IDENT **from** jobs **where** JOB\_IDENT **IN** (**select** JOB\_ID **from** employees **where** YEAR(B\_DATE)>1976 );

* Retrieve JOB information for female employees whose birth year is after 1976.

**select** JOB\_TITLE, MIN\_SALARY,MAX\_SALARY,JOB\_IDENT **from** jobs **where** JOB\_IDENT **IN** (**select** JOB\_ID **from** employees **where** YEAR(B\_DATE)>1976 **and** SEX='F' );

**Module 4 - Working with real-world data sets**

Working with Real World Data Sets OBJECTIVES

* Work with and load CSV files in a database
* Describe how attribute names in headers of CSV files map to column names in db2
* Restrict rows in a query to sample data in a table.

CSV= comma-separated values. It can actually be separated by semi columns “;”

We use example: DOGS-CSV

|  |  |  |  |
| --- | --- | --- | --- |
|  | ID | NAME\_OF\_DOG | BREED |
| 1 | 1 | Wolfie | German sheppard |
| 2 | 2 | Fluffy | pomeranian |
| 3 | 3 | Huggy | Labrador |

CSV FILES have HEADERS, you may activate it indb2 in the “header in first row” button

**Querying column names with mixed (upper and lower) cases**

If in a CSV file the name of the column contains spaces, for example, the "Name <space> of <space> Dog", by default, the database may map them to underscores that is, "NAME <underscore> OF <underscore> DOG“ Other special characters like parentheses or brackets may also get mapped to underscores. Therefore, when you write a query, ensure you use uppercase within quotes and substitute special characters with underscores as shown in this example.

|  |
| --- |
| Example |
| Select “ID”, NAME\_OF\_DOG”,  “BREED\_DOMINANT\_BREED\_IF\_NOT\_SURE\_PURE\_BREED\_”  From DOGS; |

Finally, it's also important to notice the trailing underscore after “breed" that is put in place of the closing bracket.

**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:*

So, how would you restrict the number of rows retrieved? A table may contain thousands or millions of rows, but you may only want to sample certain data, or look at just a few rows to see what kind of data the table contains.

You may be tempted to just do "select (star) from tablename" to retrieve the results. But doing so may take a long time for a query to run. Instead, you can restrict the result set by using the LIMIT clause. For example, use the following query to retrieve just the first three rows in a table called census data:

|  |
| --- |
| Example |
| Select \* from census\_data LIMIT 3 |

LESSONS:

* CSV is common data format for loading tables.
* CSV files may contain a first header row that needs to be treated differently from data rows
* Many RDBMes like db2 by default map mixed case columng names to uppercase and spaces and special characters to underscores
* Data in a table can be sampled using the LIMIT clause in a SELECT query to retrieve a few rows.

**GETTING TABLE AND COLUMN DETAILS**

* identify where database object metadata is kept + where metadata for database objects is stored.
* Retrieve the list of tables in a database and their properties.
* access information about each column in a table.

Now, how would you get a list of the tables in a database? Sometimes your database may contain several tables and you may not remember the correct names.

Database systems typically contain system or catalog tables where you can query the list of objects (such as tables) and get their properties. This information about database objects is also called **metadata.**

In Db2 this catalog is called **“SYSCAT.TABLES.”.** In SQL Server it is **“INFORMATION\_SCHEMA.TABLES”** And in Oracle it is **“ALL\_TABLES” or “USER\_TABLES.”**

|  |  |
| --- | --- |
| Db2 | SYSCAT.TABLES |
| SQL server | INFORMATION\_SCHEMA.TABLES |
| Oracle | ALL\_TABLES or USER\_TABLES |
| MySQL | SHOW TABLES |

These special system tables can be queried using regular SQL statements as we will see next. However, in databases like MySQL you can even enter a shortcut command like "SHOW TABLES” on the MySQL Command Line Prompt to get the list of tables. To get a list of tables in a Db2 database you can run the following query:

|  |
| --- |
| Formula |
| Select \* from syscat.tables |

This select statement will return too many tables, including system tables, so it is better to filter the result set by specifying a user schema name as shown here:

|  |
| --- |
| Formula |
| Select TABSCHEMA, TABNAME, CREATE\_TIME  From syscat.tables  Where TABSCHEMA = “ABC12345” |

Ensure you replace ‘ABC12345’ with your own Db2 username or the schema you want to get details for.

When you do a select \* from syscat.tables you get all the properties of the tables. Sometimes **you may be interested in specific properties** such as creation time. Let’s say you have created several tables with similar names, you need to query this:

|  |
| --- |
| Example |
| Select TABSCHEMA, TABNAME, CREATE\_TIME  From syscat.tables  Where TABSCHEMA = “PYV10949” |

The output will contain the schema name, table name, and creation time of all tables in your schema.

**Access info in columns**

Now, let’s talk about how to get a list of columns in a table. If you can't recall the exact name of a column, for example, if you are unsure whether the column name had an underscore as a separator, in Db2 you can issue a query like the one shown here:

|  |
| --- |
| Example |
| Select \* from syscat.columns  Where tabname = “DOGS” |

In MySQL you can simply run the command "SHOW COLUMNS FROM DOGS."

Or there could be a scenario where you may want to know specific properties, like the data type, and the length of the data type. In Db2, you can issue a statement like:

|  |
| --- |
| To obtain specific column properties: |
| Select distinct (name), coltype, length  From sysibm.syscolumns  Where tbname = “DOGS” |

Otro ejemplo:

|  |
| --- |
| Example |
| Select distinct(name), coltype, length  From sysibm.syscolumns  Where tbname = “Chicago\_crime\_data” |
| Resultado: |
| NAME // coltype // length |
| Arrest VARCHAR 5  Etc etc… |

En resumen:

* Metadata about database objects is stored in System Catalog or Information Schema
* You can retrieve list of tables in Db2 by querying SYSCAT-TABLES
* You can retrieve a list of columns and their properties in a Db2 table by querying SYSCAT.COLUMNS or SYSIBM.SYSCOLUMNS