SQL Data with Bara (14 Sep'2024 - 16 Sep'2024)

Overview of course

- **Data** - Data are facts and statistics that are stored or flow over a network
- **Information** - data can be transferred, processed or structured into a new form called information
- **Database** - A collection of structures and related data organised to be easily access and managed

Difference data bases

- Relational SQL Databases
- NoSQL Databases
- Distributed DB
- Cloud DB
- Warehouse DB

Difference Between SQL and NOSQL

SQL	NOSQL
- Relational DB - Structured in table - Have relational between table - Easy to access	- No Relational DB - Uses Hierarchical model - Scalable in horizontal and vertical - Data is accessed through key value, column value, Graph structure

- To manage databases we use DBMS Software (There are 380 DBMS we have)
- We use SQL language to speak with DBMS to make any manipulation on the databases

Five Reasons learn SQL

- SQL is the oldest language around 47 years old
- There are 700+ Computer language still stands on top
- SQL is the most used language in all companies because the data is stored in relational model
- Demand for SQL is high in the market

SQL Commands (there are 12 SQL Commands and 900 SQL Keywords)

We will be using the only most used keywords in this tutorial

DDL **Data Definition language**	**DQL** **Data Query Language**	**DML** **Data Manipulation Language**	**DCL** **Data Control Language**	**TCL** **Transaction Control Language**
- CREATE - DROP - ALTER	-SELECT	- INSERT - DELETE - UPDATE	- GRANT - REVOKE	- COMMIT - ROLLBACK - SAVE POINT

SQL Basic Elements

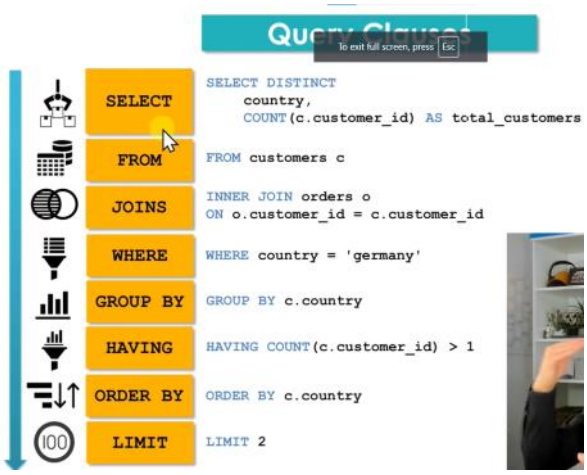
Keywords (internally defined)

```
SELECT first_name, Last_name
FROM Customer
WHERE country = 'India'
      And Score < 500
```

Golden Rules of SQL Statement

- Always add a new lines for each Column/Keyword
- Add Comment line for better understanding of the query
- Uppercase all KEYWORDS
- Add write spaces

Order of SQL Clauses



DQL - SELECT Keyword

- **SELECT** Keyword helps you in retrieving data from the data base

  SELECT first_name, Last_name

  FROM Customers

  It retrieves first name and last name from the customer table

  SELECT * FROM Customer (* will pull everything from the table)

DISTINCT -

- To fetch unique data sets from the table and remove duplicates
- DISTINCT keyword always come after SELECT keyword
- It checks on Column data sets and fetch distinct values and remove duplicates

SELECT DISTINCT
country
FROM Customers

To fetch unique country and remove duplicates and

ORDER BY-

- ORDER BY keyword is used to sort the data in the table
- And it has two difference types of sorting techniques
  - **ASC** - Ascending order - Smallest first
  - **DESC**-Descending Order - Highest First
- ORDER BY works on column wise and sort the data based on Query
- By Default in SQL we have ASC when we mention ORDER BY keyword - sort the data from smallest first

DESC	**ASC**
SELECT *FROM Customers ORDER BY country DESC	SELECT *FROM Customers ORDER BY country ASC

FILTERING DATA

WHER Keyword - filters rows based on specific condition

*SELECT * FROM customers*
WHERE Country = 'Germany'

It filter only Germany country Rows

WHERE is a powerful Operator in SQL, It plays crucial role in data analysis

We have two kinds of Operators in WHERE keyword

COMPARISION	**LOGICAL**
It compared TWO values and return the result • = Equal • !=, <> Not Equal • > Greater Than • < Less Than • => Greater Than or Equal to • <= Less than or Equal to	It compares more than two conditions and return result • **AND** - Return TRUE if both conditions are TRUE *SELECT * FROM customers* *WHERE Country = 'Germany' AND score > 400* • **OR** - Return TRUE if one of Condition is TRUE *SELECT* **FROM customers* *WHERE country = 'Germany'* *OR score < 400*

- **NOT** - reverse the result of any Boolean operator

SELECT
** FROM Customers*
WHERE NOT score < 400

BETWEEN - Return TRUE if a value fall within a specific range

- We use AND Operator in between to give range

SELECT
**FROM customers*
WHERE score BETWEEN 100 and 500

- Instead of using BETWEEN operator, we can use two Logical conditions to get the data

SELECT
** FROM customers*
WHERE score >=100 AND score <= 500

IN - Return TRUE if a value is in a list of values

- Have to specify the values in parenthesis

SELECT
**FROM customers*
WHERE customer_id IN (1,4,5)

LIKE - Return TRUE if a value matched with PATTERN

- There are two tool **%(Percentage)** and **_(underscore)**
- Case sensitive - need to be careful while mentioning in query
- **%(Percentage)** - it matches everything and return the result
- **_(underscore)** - It matched exactly one character and return the result

Find name Begin with 'M' *SELECT* ** FROM customers* *WHERE first_name LIKE 'M%'*	M%
Fine name end with 'n; SELECT * FROM customers WHERE first_name LIKE '%n'	%n
Fine name containing the 'r' *SELECT* ** FROM customers* *WHERE first_name LIKE '%r%'*	%r%
Find names containing the 'r' at 3rd Position *SELECT* **FROM customers* *WHERE first_name LIKE '__r%'*	_ _r%

SQL Aliases - it helps to create a short aliases name for a table or column

- **To give surname for the table**

SELECT
c.customer_id
FROM customers AS c

- **To give surname for the Column and table**

SELECT
c.customer_id AS CID
FROM customers AS c

- it will rename in the script and result and it won't change in the date case

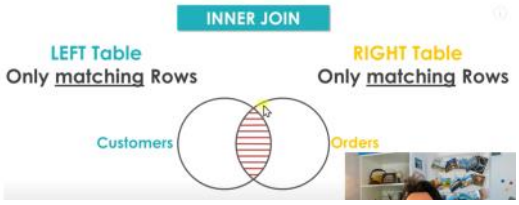
JOINS

- It helps you to work with multiple tables
- To create Join key between two tables
- Four types of Join **(INNER JOIN, LEFT JOIN , RIGHT JOIN, FULL_JOIN)**
- In order to combine two tables we have to define **JOIN KEY (in both table we should have a common columns)**
- Have to define type of join

- In Joins we will have RIGHTand LEFT Tables

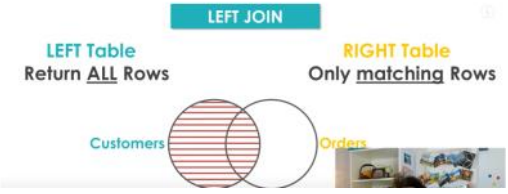
INNER JOIN - It returns only those rows that have a match in both joined tables
- While creating inner join we should have unique columns between table
- Give a nick names for the table to pull data between tables
- Inner join works when there is same column between two tables so we can create **JOIN KEY**
- To create a key between two tables we use **ON** keyword

```
SELECT
   c.customer_id,
   c.first_name,
   o.order_id,
   o.quantity
FROM customers AS c
INNER JOIN orders AS o
ON c.customer_id          =          o.customer_id
```



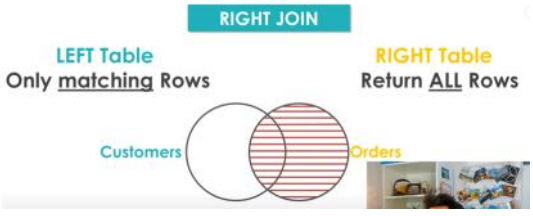
LEFT JOIN - It returns all the rows from the left + matching in Right Table
- it pulls everything from the left table and pull only matching from right table

- LEFTjoin works when there is same column between two tables so we can create **JOIN KEY**
- To create a key between two tables we use **ON** keyword




```
SELECT
        c.customer_id,
   c.first_name,
   o.order_id,
   o.quantity
FROM customers AS c
LEFT JOIN orders AS o
ON c.customer_id = o.customer_id
```

RIGHT JOIN - It returns all the rows from the Right + matching in Left Table
- It pull everything from the right table and pulls only matching from left table
- RIGHT join works when there is same column between two tables so we can create **JOIN KEY**
- To create a key between two tables we use **ON** keyword

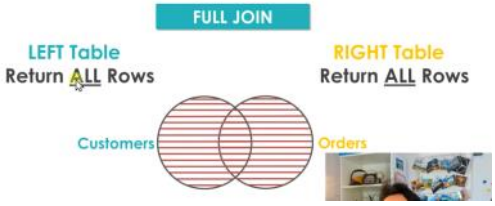



```
SELECT
        c.customer_id,
   c.first_name,
   o.order_id,
   o.quantity
FROM customers AS c
RIGHT JOIN orders AS o
ON c.customer_id = o.customer_id
```

FULL JOIN - It returns all the rows from the Right & Left Table
- Most of the DBMS SW wont support FULL JOIN like in mysql wont support

- We will be using LEFT and RIGHT Join and UNION keyword to combine both tables in mysql to pull data from the two tables

```
SELECT
        c.customer_id,
    c.first_name,
    o.order_id,
        o.quantity
FROM customers AS c
RIGHT JOIN orders AS o
ON c.customer_id = o.customer_id
UNION
SELECT
        c.customer_id,
    c.first_name,
    o.order_id,
        o.quantity
FROM customers AS c
LEFT JOIN orders AS o
ON c.customer_id = o.customer_id
```



UNION - UNION helps you to combine two tables and it is important role in collating two table to pull everything

- It pull all the data from two tables by excluding duplicate data from the two table which means it combines rows in both table those are same
- It will select the column headers from the LEFT table
- We should be have same columns and data between two tables
- In UNIONS the data types of columns must match (if there is any mismatch-Query will not run)
- We can give nick names in the first table query but cant give nick names in the second table query
- There are two types of UNION
    - **UNION ALL** - combine the rows without removing duplicates
    - **UNION** - combine the rows and removing duplicates

UNION - combine the rows and removing duplicates	**UNION ALL - combine the rows without removing duplicates**
SELECT first_name AS per_first_Name, last_name AS per_Last_name, country AS per_Country_Name FROM customers UNION SELECT first_name, last_name, emp_country FROM employees	SELECT first_name, last_name, country FROM customers UNION ALL SELECT first_name, last_name, emp_country FROM employees

AGGREGATE FUNCTION - SQL have set of aggregate functions

⇒ **COUNT()**	**- Returns the number of rows in result set**
⇒ **SUM()**	**- Return the sum of value**
⇒ **AVG()**	**- Returns the average of values**
⇒ **MAX()**	**- Return the Maximum Value**
⇒ **MIN()**	**- Return the Minimum Value**

⇒ **COUNT()**	**- Returns the number of rows in result set**

- Its just count the number of records in the table
- COUNT can use it on numeric and alpha data
- And it ignores the blank data in the column
- We can count the whole table COUNT(*) or column COUNT(first_name)
- We can give nick names using as *COUNT(score) AS Total_Score*

SELECT COUNT(score) AS Total_Score
FROM customers

⇒ **SUM()**	- **Return the sum of value**

- Its sum the value and return the result
- It works on the column where columns have numeric data
- Works only with numeric columns
- NULLS or blank data treated as ZERO
- We can give nick names using as SUM(quantity) AS Total_sum_Quantity

SELECT SUM(quantity) AS Total_sum_Quantity
FROM orders

⇒ **AVG()**	- **Returns the average of values**

- Return the average values in the column
- Works only with numeric columns
- NULL or Blanks are ignored
- We can give nick names using as AVG(score) AS Average_Score

SELECT AVG(score) AS Average_Score
FROM Customers

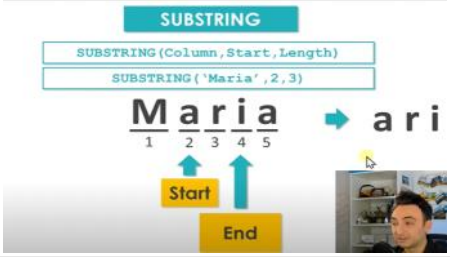
⇒ **MAX()**	- **Return the Maximum Value** - We can use it on date as well SELECT MAX(score) AS Maximum_Score FROM Customers
⇒ **MIN()**	- **Return the Minimum Value** - We can use it on date as well SELECT MIN(score) AS MINIMUM_VALUE FROM customers

STRING FUNCTIONS

- To query with Alpha data in the columns
- We have the below string function to analyze Alpha data in the columns

CONCAT()	Return a string by concatenating two or more string values *SELECT CONCAT(first_name, '-', Last_name) AS Full_Name* *FROM customers*
LOWER()	Converts a string to lowercase *SELECT LOWER (first_name) AS LOWER_CASE* *FROM customers*
UPPER()	Convers a string to uppercase *SELECT UPPER(first_name) AS Upper_Case* *FROM customers*
TRIM()	Remove leading and trailing spaces from a string In TRIM we have two types - LTRIM Left Trim - Remove spaces in the left LTRIM use to remove space in the left_ TRILOK_(Space in the right to remove use RTRIM - RTRIM Right Trim - Remove unnecessary spaces in the right - TRIM() to remove from the both of the side *SELECT TRIM(last_name) AS CLEAR_NAMEFROM* *FROM customers*
LENGTH()	Returns the length of string - By using LENGTH and TRIM SELECT last_name, TRIM(last_name) AS CLEAR_NAME, LENGTH(last_name) AS Length_Name, LENGTH(TRIM(last_name)) AS Clean_Length_last_Name FROM customers
SUBSTRING()	Return a substring from strings - Each Character has a position in the table **SUBTRIM(Column, Start, length)** • **COLUMN** tell from which column name to check,

- **START** will check from the number of character
- **LENGTH** will give you the result from START character to LENGTH of the character



SQL GROUP BY CLAUSE

- Grouping rows based on a column values
- GROUP BY keyword helps in providing the unique data from the table
- It boost with the other keyword like count, aggregate functions

```
SELECT
        COUNT(*) AS TOTAL_Customers,
    country
    FROM customers
    GROUP BY country
```

- We can use ORDER BY to Ascending and descending - it comes after GROUP BY

```
SELECT
        COUNT(*) AS TOTAL_Customers,
    country
    FROM customers
    GROUP BY country
    ORDER BY COUNT(*) DESC
```

- We can use MAX to check the maximum value

```
SELECT
        MAX(score) AS MAXIMUM_SCORE,
    country
FROM customers
GROUP BY country
ORDER BY MAX(score) ASC
```

HAVING CLAUSE

- It works when you wants to specific which data set you required
- Filters the groups returned by GROUP BY
- Need to use after GROUP BY
- WHERE Filter works on columns wise (Filters rows before any groupings take place)
- Filter values after they have been groups
- HAVING works only with GROUP BY Clause

USING HAVING to get greater than 1

```
SELECT
COUNT(*) AS TOTAL_CUSTOMERS,
country
FROM customers
GROUP BY country
HAVING COUNT(*) >1
```

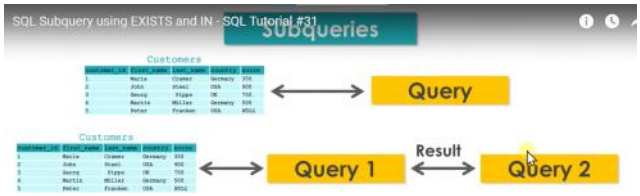
USING WHERE to exclude specific from the column

```
SELECT
COUNT(*) AS TOTAL_CUSTOMERS,
country
FROM customers
WHERE country != 'USA'
GROUP BY country
HAVING COUNT(*) >1
```

SUBQUERIES

- Using Nested Queries and works on Query 2 result to Query 1
- We use EXISTS and IN

IN - USING IN keyword and adding sub queries



SELECT
** FROM orders*
WHERE customer_id IN
(SELECT customer_id
FROM customers
WHERE score > 500)

EXISTS

- Use EXISTS instead of IN for Large tables
- You will get better performance with EXISTS

*SELECT * FROM orders AS o*
WHERE EXISTS (
SELECT 1
FROM customers AS c
WHERE c.customer_id = o.customer_id
AND score > 500
)

DML - DATA MANIPULATION LANGUAGE

- To change the data inside the database
- We have DML _data manipulation language and there are three major keywords

▸ **INSERT**	- To add the rows of data into the table
▸ **UPDATE**	-To modify the values of an existing rows of table
▸ **DELETE**	- To delete from rows in the table

- In order to make any changes in the table we should understand the structure of table
- Use **DESCRIBE** keyword to understand the structure of table(Alpha, Numeric,, Primary keys, Null, Integers, Float, character restrictions……)

▸ INSERT - To add the rows of data into the table

- **INSERT INTO** - where to add the new row
- **VALUE** - helps you to add the data in every column
- For Primary keys column, always use default, because it should be unique.

  INSERT INTO customers
  VALUE(DEFAULT, 'Trilok', 'Mohithe', 'India', 0)
- Incase if you want to specify the columns name (we can add columns attributes in the query)
- In the below query first_name tells to add first_name as Laxman and Last_name as Yekabote

  INSERT INTO customers
  (first_name, Last_name)
  VALUE('Laxman','Yekabote')
- For NULL values, we can add NULL or 0 zero

▸ UPDATE -To modify the values of an existing rows of table

- **UPDATE** - keyword to modify the data
- **SET** - helps to change the specific row
- When we are modifying data, table should have a primary to know where to make changes
- To do that **WHERE** keyword to give address of Primary key row number

  UPDATE customers
  SET country = 'Indonesia', score = 740
  WHERE customer_id = 2

▸ DELETE - To delete from rows in the table

- **DELETE -** It simple but dangerous to use in case if you give wrong instructions
- **DELETE** without using **WHERE** keyword will delete all rows from a table

  DELETE FROM customers
  WHERE customer_id = 7

Using **IN** to delete multiple rows

```
DELETE FROM customers
WHERE customer_id in (5,6)
```

DELETE keyword would take very long time to delete all row by row

```
DELETE FROM customers
```

Another keyword **TRUNCATE** we can use to delete the table in one shot

```
TRUNCATE customers
```

DDL - DATA DEFINATION LANGUAGE

- DDL will help you in changing the structure of data base

- **CREATE**	
- **DROP**	
- **ALTER**	

COLUMN Definition

COLUMN Name	Data Type	Constraints
Could be anything	INT	PRIMARY KEY
	VARCHAR	NOT NULL
	DATE	UNIQUE
	CHAR	DEFAULT
	……	…..

CREATE

- Create a table by using CREATE and adding data by using column name, data types and constraints

```
CREATE TABLE db_sql_tutorial.Person(
Id INT PRIMARY KEY AUTO_INCREMENT,
Full_Name VARCHAR(50) NOT NULL,
DOB DATE,
Phone VARCHAR(15) UNIQUE NOT NULL
)
```

- First thing is to add keyword as CREATE TABLE
- Mention data_base name where you are creating and link new table name to data base
- Add column headers and define data types and contraints
  - PRIMARY KEY (creating unique value)
  - AUTO_INCREMENT (value will generate auto and unique)
  - VARCHAR() defines how many characters required
  - DATE - date to insert
  - UNIQUE - to have unique data in the columns to avoid duplicates
  - NOT NULL - should not be blank
  - NULL - can be blank

ALTER TABLE

To add any additional columns

```
ALTER TABLE person
ADD Email VARCHAR(20) NOT NULL
```

DROP TABLE

To delete a table we use DROP keyword

```
DROP TABLE person
```