TOPICS COVERED

1) Order By
2) Limit
3) Aliases
4) Aggregate Functions
   (MIN(), MAX(), COUNT(), SUM(), AVG()
   ROUND())
5) String manipulation functions
   (LENGTH(), UPPER(), LOWER(), SUBSTRING(),
   MID(), LOCATE(), REPLACE(), TRIM(),
   CONCAT())
6) DATE functions
   (NOW(), CURDATE(), CURTIME(), DATE(),
   YEAR(), MONTH(), DAY(), EXTRACT(),
   DATE_ADD(), DATE_SUB(), DATEDIFF(),
   DATE_FORMAT())
7) NULL functions
   (ISNULL(), ISNOTNULL(), COALESCE(),
   IFNULL(), NULLIF())
8) NUMBER functions
   (ABS(), CEIL(), FLOOR(), ROUND(), MOD(),
   POWER(), SQRT(), RAND())

ORDER BY Keyword:

The ORDER BY keyword is used to sort the result-set in ascending or descending order

* By default it is ascending order. If we want it in descending order, we use the DESC keyword.

Syntax:

```
SELECT Column1, Column2, ....
FROM <table_name>
ORDER BY Column1, Column2,... ASC|DESC;
```

Example: ASC Example

```
SELECT *
FROM Customers
ORDER BY Country;
```

DESC Example:

```
SELECT *
FROM Customers
ORDER BY Country DESC;
```

ORDER BY Multiple Columns Example:

```
SELECT *
FROM Customers
ORDER BY Country, City;
```

(OR)

```
SELECT *
FROM Customers
ORDER BY Country ASC, City DESC;
```

SQL LIMIT Clause:

The SQL LIMIT clause is used to specify the number of records to return.

* In MS Access we use TOP & ORACLE uses ROWNUM.

Syntax:

```
SELECT column1, column2, ...
FROM <table_name>
LIMIT number;
```

Example

```
SELECT *
FROM Customers
LIMIT 3;
```

(or)

```
SELECT *
FROM Customers
WHERE Country = 'India'
LIMIT 3;
```

Equivalent Examples in MS ACCESS & ORACLE:

MS ACCESS Example:

```
SELECT TOP 3 *
FROM Customers
WHERE Country = 'India';
```

ORACLE Example:

```
SELECT *
FROM Customers
WHERE Country = 'India' AND ROWNUM <= 3;
```

SQL Aliases:

SQL aliases are used to give a table, or a column in a table, a temporary name.

* Aliases are often used to make column names more readable.
* An alias only exists for the duration of the query

Alias Column Syntax:

```
SELECT column_name AS alias_name
FROM table_name;
```

Alias Table Syntax:

```
SELECT column1, column2, ....
FROM table_name AS alias_name;
```

Examples:

```
SELECT ID AS Customer_ID,
       Custmer_name AS Customer
FROM Customers;
```

* If Alias name consists of space, it needs to be enclosed in double quotes or square brackets

```
SELECT ID AS Customer_ID,
       Custmer_name AS [contact Person]
FROM Customers;
```

(or)

```
SELECT ID AS Customer_ID,
       Custmer_name AS "contact Person"
FROM Customers;
```

```
SELECT O.OrderID, O.OrderDate, C.Customer_name
FROM Customers AS C, Orders AS O
WHERE C.Customer_name = "xxx" AND
      C.Customer_ID = O.customer_ID;
```

* Aliases can be useful when:

- There are more than one table involved in a query
- Functions are used in the query
- Column names are big and not very readable.
- Two or more columns are combined together.

SQL Aggregate functions:

SQL MIN() and MAX() Functions:

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

MIN() Syntax:

```
SELECT MIN(column_name)
From <table_name>
WHERE condition;
```

MAX() syntax:

```
SELECT MAX(column_name)
From <table_name>
WHERE condition;
```

Examples:

```
SELECT MIN(age) As min_age
From customers;
```

```
SELECT MAX(age) AS max_age
From customers;
```

SQL Count(), AVG() and SUM() Functions:

The Count() function returns the number of rows that matches a specified criteria.

The AVG() function returns the average value of the numerical column.

The SUM() function returns the total sum of a numerical column.

Count() Syntax:

```
SELECT Count(Column_name)
FROM <table_name>
WHERE condition;
```

AVG() Syntax:

```
SELECT AVG(Column_name)
FROM <table_name>
WHERE condition;
```

SUM() Syntax:

```
SELECT SUM(Column_name)
FROM <table_name>
WHERE condition;
```

Examples:

```
SELECT COUNT(ID)
FROM Customers
WHERE ID>2;

SELECT AVG(age)
FROM Customers
WHERE Country = 'India';

Select SUM(Salary)
FROM Customers
WHERE Country = 'India';
```

SQL ROUND( ) Function :

The ROUND( ) function is used to round a numeric field to the number of decimals specified.

Syntax :

```
SELECT ROUND(column_name, decimals)
FROM <table_name);
```

* Here column_name represents the field to round.
* decimals represents the number of decimals to be returned.

Example :

```
SELECT ProductName,
    ROUND(Prices,0) As Rounded price
FROM Products.
```

SQL String manipulation functions:

In MySQL there are several manipulation functions available that allow us to modify and manipulate string values.

SQL LENGTH() function:

The LEN() function returns the length of the value in a text field.

Syntax:

```
SELECT LENGTH (column_name)
FROM <table_name>;
```

Example:

```
SELECT Customer_name,
  LENGTH (Address) As length_of_address
FROM Customers;
```

SQL LOWER( ), UPPER( ) functions:

The LOWER( ) function converts a string to lowercase.

The UPPER( ) function converts a string to uppercase.

LOWER Syntax:

```
SELECT LOWER(column_name)
FROM <table_name>;
```

UPPER Syntax:

```
SELECT UPPER(Column_name)
FROM <table_name>;
```

Example:

```
SELECT UPPER(LastName) AS upper_lastname,
   LOWER(FirstName) As lower_firstname
FROM Customers;
```

SQL SUBSTRING( ), MID( ) functions:

The SUBSTRING() function extracts characters from a text field.

The MID() function also extracts characters from a text field. Here it is not mandatory to mention length. This is MYSQL specific command.

SUBSTRING( ) syntax:

```
SELECT SUBSTRING(column_name,
    start_position, length) AS some_name
FROM <table_name>;
```

MID() syntax

```
SELECT MID(column_name,
    start_position, length) AS some-name
FROM <table_name>;
```

* here length is not mandatory, if unspecified it retreives until the end.

Example:

```
SELECT SUBSTRING(city, 1, 4) AS Shortcity
FROM Customers;

SELECT MID (city, 1, 4) AS Shortcity
FROM Customers;
```

SQL LOCATE(), REPLACE(), TRIM() functions:

The LOCATE() function finds a substring with in a text field.

The REPLACE() function replaces all the occurances of the 'search' string with the 'replace' string within the column.

The TRIM() function removes leading and trailing spaces (or other specified characters) from the column.

LOCATE() syntax:

```
SELECT LOCATE(Substring,
        Column_name, start) AS Some_name
FROM <table_name>;
```

* start parameter is optional, default is 1, specifies the position to start search from.

Example:

```
SELECT LOCATE ('son', full_name, 5)
        As position
FROM Customers;
```

start position optional

```
SELECT LOCATE ('son', full_name)
        As position
FROM Customers;
```

REPLACE ( ) Syntax:

```
SELECT REPLACE ( column_name,
        search, replace) As some_name
FROM <table_name>;
```

Example:

```
SELECT REPLACE ( description, 'old', 'new')
FROM products;
```

(or)

```
UPDATE products
SET description = REPLACE (description,
        'old', 'new');
```

TRIM ( ) syntax:

```
SELECT TRIM (character FROM column_name)
        AS some_name
FROM <table_name>;
```

* default character to trim is space if it is not mentioned.

Example:

```
SELECT TRIM ('_' FROM product_name)
        AS trimmed_name
FROM products
```

```
SELECT TRIM (product_name)
        AS trimmed_name
FROM products
```

↙ if we need to remove spaces

SQL CONCAT( ) function:

The CONCAT( ) function is used to concatenate two different text fields (columns) in to one text field.

Syntax:

```
SELECT CONCAT(column1, seperator,
       column2,...) As some_name
FROM <table_name>;
```

Example

```
SELECT CONCAT(FirstName,' ', LastName)
       As full_name
FROM Customers;
```

SQL Dates:

* The most difficult part when working with dates is to be sure that the format of the date we are trying to insert matches the format of date column in the database.

* As long as the data contains only the date portion, queries will work as expected. However, if time portion is involved, it gets more complicated.

SQL Date Datatypes:

MYSQL comes with the following data types for storing a date or a date/time value in the database:

1) DATE - YYYY-MM-DD
2) TIME - HH:MI:SS
3) DATETIME - YYYY-MM-DD HH:MI:SS
4) TIMESTAMP - YYYY-MM-DD HH:MI:SS
5) YEAR - YYYY or YY

MYSQL NOW() Function:

The NOW() returns the current date and time.

Syntax:

```
NOW()
```

Output:

```
Current DATETIME
    YYYY-MM-DD HH:MI:SS
```

CURDATE() function:

The CURDATE() returns the current date.

Syntax:

```
CURDATE()
```

Output:

```
Current date
    YYYY-MM-DD
```

CURTIME() function:

The CURTIME() returns the current time

Syntax:

```
CURTIME()
```

Output:

Current time

HH:MI:SS

DATE() function:

The DATE() extracts the date part of date or date/time expression.

* can be used with NOW(), CURDATE() functions.

Syntax:

DATE(date)

* where date is a valid date expression

Output: date part of expression

YYYY-MM-DD.

YEAR() function:

The YEAR() extracts the year part of date or date/time expression.

* can be used with NOW(), CURDATE() functions.

Syntax:

YEAR(date)

* where date is a valid date expression

Output: year part of expression

YYYY .

MONTH ( ) function:

The MONTH ( ) extracts the month part of date or date/time expression.

* can be used with NOW( ), CURDATE( ) functions.

Syntax:

MONTH (date)

* where date is a valid date expression

Output: month part of expression

MM

DAY ( ) function:

The DAY ( ) extracts the day part of date or date/time expression.

* can be used with NOW( ), CURDATE( ) functions.

Syntax:

```
DAY (date)
```

* where date is a valid date expression

Output: day part of expression

DD

Usage:

```
SELECT NOW() As current_datetime,
    CURDATE( ) AS current_date,
    CURTIME ( ) As current_time,
    DATE (NOW ( )) As extracted_date,
    DAY (NOW( )) As extracted_day,
    MONTH (NOW( )) As extracted_month,
    YEAR(NOW( )) As extracted_year;
```

MYSQL EXTRACT( ) function:

The EXTRACT( ) function is used to extract a specific part (such as year, month, day, hour etc) from a date or date/time expression.

Syntax:

```
EXTRACT (unit FROM date_expression)
```

* Unit is part to extract - 'YEAR', 'MONTH', 'DAY', 'HOUR', 'MINUTE', 'SECOND' etc.
* date_expression is the date or datetime value from which to extract the specified part.

Example:

```
SELECT EXTRACT (YEAR FROM '2023-06-03')
    As extracted_year,
  EXTRACT (MONTH FROM '2023-06-03')
    As extracted_month,
  EXTRACT ( DAY FROM '2023-06-03')
    As extracted_day;
```

* we have many supported units in EXTRACT( ) function.

MYSQL DATE_ADD( ) function:

The DATE_ADD( ) function is used to add a specified interval to a date or datetime expression. It allows us to add years, months, days, hours, minutes, seconds, etc, to a given date.

Syntax:

DATE_ADD(date_expression, INTERVAL value unit)

* 'date_expression' is the date or datetime value to which the interval should be added.
* 'value' specifies the number of units to add.
* 'unit' specifies the unit of time to add such as 'YEAR', 'MONTH', 'DAY', 'HOUR', 'MINUTE', 'SECOND', etc.

Example:

```
SELECT DATE_ADD('2023-06-03', INTERVAL 1
MONTH) AS added_date;
```

MYSQL DATE_SUB( ) function:

The DATE_SUB( ) function is used to subtract a specified interval to a date or datetime expression. It allows us to subtract years, months, days, hours, minutes, seconds, etc, to a given date.

Syntax:

DATE_SUB (date_expression, INTERVAL value unit)

* 'date_expression' is the date or datetime value to which the interval should be subtracted
* 'value' specifies the number of units to subtracted
* 'unit' specifies the unit of time to subtract such as 'YEAR', 'MONTH', 'DAY', 'HOUR', 'MINUTE', 'SECOND', etc.

Example:

```
SELECT DATE_SUB('2023-06-03', INTERVAL 1
        WEEK) AS Substracted_date;
```

MySQL DATEDIFF() Function:

The DATEDIFF() function is used to calculate the difference in days between the two dates. It returns the number of days between a given start date and end date.

Syntax:

DATEDIFF (end-date, start-date)

* 'end_date' is the later date or datetime value
* 'start_date' is the earlier date or datetime value.

Example:

```
SELECT DATEDIFF('2023-06-10', '2023-06-03')
        AS date_difference;
```

Output:

Output is 7.

(or)

```
SELECT DATEDIFF(end-date, start_date)
        AS date-difference
FROM sales;
```

* New column is created with difference in the dates.

MYSQL DATE_FORMAT( ) function:

The DATE_FORMAT() function is used to format a date or datetime value in to a specific format. It allows us to convert a date or datetime value to a formatted string representation.

Syntax:

DATE_FORMAT(date_value, format_string)

* `date_value' is the date or datetime value that we want to format.
* `format_string' specifies the desired format for the output.

The `format_string' parameter follows a specific format pattern that includes various placeholders for different parts of the date or time. Below are some commonly used specifiers:

* '%Y': year with 4 digits (eg 2023)
* '%y': year with 2 digits (eg 23)
* '%m': month with 2 digits (eg 06)
* '%d': day with 2digits (eg 03)
* '%H': Hour in 24hour format as 2 digit (eg 17 for 5pm)
* '%i': minute as 2 digits (eg. 30)

* '%s': Second as 2 digits (eg. 45)

* '%p': AM/PM indicator (eg. AM or PM)

Example:

```
SELECT DATE_FORMAT ('2023-06-03 15:30:45',
        '%d-%m-%Y %h:%i %p) AS formatted_date;
```

Output:

03-06-2023 3:30 PM

(or)

```
SELECT DATE_FORMAT (date_column, '%Y-%m-%d')
        As formatted_date
FROM sales;
```

MySQL NULL functions:

There are several functions available to handle NULL values. Here are some commonly used NULL functions:

ISNULL: checks if a value is NULL

Usage:

```
SELECT column_name
FROM <table_name>
WHERE Column_name ISNULL;
```

Example:

```
SELECT Customer_name
FROM customers
WHERE phone_num ISNULL;
```

ISNOTNULL: checks if a value is NOT NULL

Usage:

```
SELECT column_name
FROM <table_name>
WHERE Column_name ISNOTNULL;
```

Example:

```
SELECT Product_name
FROM products
```

```
WHERE quantity ISNOTNULL;
```

COALESCE : Returns the first NON-NULL value from a list of expressions.

Usage :

```
SELECT COALESCE(expression1, expression2, ...)
        As result
FROM <table_name>;
```

* It evalutes the arguments in the order they are specified and stops at the first non-NULL value encountered.
* If all the arguments are NULL, it will return NULL.

Example :

```
SELECT employee_name, COALESCE(salary, 0)
        As adjusted_salary
From employees;
```

* This query returns the employee names and their adjusted salaries. If the salary is NULL it is replaced with 0.

IFNULL: Takes two expressions, returns first expression if not null or returns second one.

Usage:

```
SELECT IFNULL (expression1, expression2)
        As result
FROM <table_name>;
```

* It returns expression1 if it is NOT NULL, Otherwise it returns expression2
* 'IfNULL' is used when we want to replace a single NULL value with a specified value

Example:

```
SELECT IFNULL (salary, 10000)
        As adjusted_salary
FROM employees;
```

* In this example, if 'salary' column is NOT NULL it will be returned as adjusted_salary. If 'salary' is NULL, then function will return 10000 which we have given as default-value as the adjusted salary

NULLIF: Returns NULL if expressions are equal, or returns expression 1.

Usage:

```
SELECT NULLIF (expression1, expression2)
        As result
FROM <table_name>;
```

* It returns NULL if expression1 is equal to expression2, otherwise returns expression1.
* expression1, expression2 can be any expressions or columns.

Example:

```
SELECT NULLIF(5,5) As result;
```

* Since they are equal the result will be NULL.

(or)

```
SELECT name, NULLIF(salary, 5000)
        As adjusted_salary
FROM Employees;
```

* Here if the original value of salary was 5000 it will be NULL in adjusted_salary otherwise the original salary will reflect.

MySQL Number functions:

MySQL Provides various number functions that allow us to perform mathematical operations and manipulate numeric data. Here we see some commonly used number functions.

ABS() function: Returns the absolute value of a number.

Usage:

```
SELECT ABS(-10);
```

output : 10

(or)

```
SELECT ABS(Quantity) As absolute_quantity
FROM orders;
```

CEIL() function: Returns the smallest integer value greater than or equal to number.

Usage:

```
SELECT CEIL(4.2);
```

output : 5

(or)

```
SELECT CEIL(Salary) As rounded-salary
FROM employees;
```

FLOOR() function: Returns the largest integer value less than or equal to number

Usage:

```
SELECT FLOOR(4.2);
```

output: 4

(or)

```
SELECT FLOOR(Salary) As rounded-salary
FROM employees;
```

ROUND() function: Rounds a number to a specified number of decimal places.

Usage:

```
SELECT ROUND(4.567, 2);
```

output: 4.56

(or)

```
SELECT ROUND(Salary, 3) As rounded-salary
FROM employees;
```

MOD( ) function : Returns the remainder of a division operation.

Usage:

```
SELECT MOD(10,3);
```

output : 1

(or)

```
SELECT MOD(salary) AS remainder_salary
FROM employees;
```

POWER( ) function : Raises a number to the power of another number

Usage :

```
SELECT POWER(2,3);
```

output : 8

SQRT( ) function : Returns the square root of a number

Usage :

```
SELECT SQRT(16);
```

Output : 4

RAND() function: Generates a random number between 0 and 1.

Usage:

```
SELECT RAND();
```

Output: 0.84675856 7301456