SQL-Operators & Functions

Agenda

- Operators
  - Arithmetic Operators
  - COMPARISON Operators
  - LOGICAL Operators
  - SET Operators
- SQL Functions
  - Single Row Function
  - Multiple Row Function
  - Analytical Function
  - Dimension Function

SQL - Arithmetic Operator

Operator	Description
+	Adds values of Both Operands
-	Subtract right hand Operand from the left hand Operand
*	Multiply the Value of Operand
/	Divide Left hand Operand with right hand Operand
%	Divide Left hand Operand with right hand Operand and return reminder

Arithmetic Operators in SELECT

- SELECT clause can contain arithmetic operators like +, * which will perform the arithmetic operation on the retrieved data

```
SELECT employee_id, salary, salary*12
FROM employees;
```

Column Alias

- Column alias is used to display more meaningful column titles, Column alias is defined using keyword ‘AS’

```
SELECT employee_id, salary, salary*12 AS annual_salary
FROM employees;
```

- If space is required in the alias, then specify the alias name within single quote

```
SELECT employee_id, salary, salary*12 AS 'Annual Salary'
FROM employees;
```

Concatenation operator

- Concatenation is used as part of SELECT clause to combine multiple fields into a single field

```
SELECT CONCAT(first_name, ' ', last_name) AS employee_name
FROM employees;
```

- Concatenation can be performed on non character fields too. Common application of this is to combine character type fields

DISTINCT keyword

- Use of keyword DISTINCT in SELECT clause, eliminates duplicate rows in the result
- To get list of unique Job Ids of employees

```
SELECT DISTINCT job_id
FROM employees;
```

- To get list of unique department id and job ids of employees

```
SELECT DISTINCT department_id, job_id
FROM employees;
```

Table Alias

- Table alias is useful when data is retrieved from multiple columns
- A column name may exist in more then one table. This may cause confusion about source of the column. Table alias name gives ability to prefix the column name with a short alias name
- It is specified in FROM clause

```
SELECT e.employee_id, e.first_name, e.last_name, e.hire_date
FROM employees e;
```

- In the above examole ‘e’ is defined as alias for table employees
- Table alias is useful while performing JOIN operations

WHERE clause

SELECT	Specifies columns to be displayed in the result
FROM	Specifies tables from which data will be queried
WHERE	Specifies criteria for filtering records (Optional)
GROUP BY	Specifies columns used for grouping the data (Optional)
HAVING	Specifies criteria for filtering grouped data (Optional)
ORDER BY	Specifies sort order in which result is displayed (Optional)

WHERE clause

- WHERE clause is used to specify conditions that must be met by the data retrieved
- The conditions restrict the data retrieved
- It is specified after FROM clause

- To select a specific employee based on employee_id

```
SELECT employee_id, first_name, last_name
FROM employees
WHERE employee_id = 100;
```

WHERE clause (contd..)

- To select employees based on first name

```
SELECT employee_id, first_name, last_name
FROM employees
WHERE first_name = 'David';

SELECT employee_id, first_name, last_name
FROM employees
WHERE first_name = 'DAVID';
```

- The above two queries will retrieve the same rows.

SQL Comparison Operators

Operator	Description
=	It checks if two operands values are equal or not, if values are equal, then condition becomes true.
!=	It checks if two operands values are equal or not, if values are not equal, then condition becomes true.
<>	It checks if two operands values are equal or not, if values are not equal, then condition becomes true.
>	It checks if the left operand value is greater than right operand value, if yes, then the condition becomes true.
<	It checks if the left operand value is less than right operand value, if yes, then the condition becomes true.
>=	It checks if the left operand value is greater than or equal to the right operand value. If yes, then condition becomes true.
<=	It checks if the left operand value is less than or equal to the right operand value. If yes, then condition becomes true.
!<	It checks if the left operand value is not less than right operand value. If yes, then condition becomes true.
!>	It checks if the left operand value is not greater than right operand value. If yes, then condition becomes true.

SQL Logical Operators

Operator	Description
ALL	It compares a value to all values in another value set.
AND	It allows the existence of multiple conditions in a SQL statement.
ANY	It compares the values in the list according to the condition.
BETWEEN	It is used to search for values that are within a range.
IN	It compares a value to that specified list value.
NOT	It reverses the meaning of any logical operator.
OR	It combines multiple conditions in SQL statement.
EXIST	It is used to search for the presence of a row in a specified table.
LIKE	It compares a value to similar values using wildcard operator.

WHERE clause (contd..)

- To select employees who joined before 1st Jan 2000

```
SELECT employee_id, first_name, last_name, hire_date
FROM employees
WHERE hire_date < '2000-01-01';
```

- To select employees who work for departments other than department_id 100

```
SELECT employee_id, first_name, last_name, department_id
FROM employees
WHERE department_id <> 100;
```

- The symbol ‘<>’ and ‘!=’ both indicate ‘not equal to’

WHERE clause (contd..) – AND operator

- To select employees are part of department 90 and have annual salary more than 100,000

```
SELECT employee_id, first_name, last_name, department_id,
      salary as monthly_Salary, salary*12 as annual_salary
FROM employees
WHERE department_id = 90
   AND salary * 12 > 10000;
```

- To select employees who belong to department_id 60 and have salary between 9000 & 10000 (both inclusive)

```
SELECT employee_id, first_name, last_name, department_id, salary
FROM employees
WHERE department_id = 60
      AND salary >= 9000
      AND salary <= 10000;
```

WHERE clause (contd..) – BETWEEN operator

- To select employees who belong to department_id 60 and have salary between 9000 & 10000 (both inclusive)
- This also be achieved using BETWEEN operator

```
SELECT employee_id, first_name, last_name, department_id,  salary
FROM employees
WHERE department_id = 60
  AND salary BETWEEN 9000 AND 10000;
```

WHERE clause (contd..) – OR operator

- To select employees who belong to either of the department 60 or 80 and have salary between 9000 & 10000 (both inclusive)

```
SELECT employee_id, first_name, last_name,
       department_id, salary
FROM employees
WHERE (department_id = 60 OR department_id = 80)
       AND salary BETWEEN 9000 AND 10000;
```

- NOTE: Use parenthesis for appropriate precedence of operators. In absence of parenthesis, operator AND takes precedence over OR operator

- To select employees who belong to either of the department 20, 30 or 60

```
SELECT employee_id, first_name, last_name, department_id
FROM employees
WHERE department_id = 20
   OR department_id = 30
   OR department_id = 60;
```

WHERE clause (contd..) – IN operator

- IN operator is used for checking if a column value is part of a set of values
- To select employees who belong to either of the departments 20, 30 or 60

```
SELECT employee_id, first_name, last_name, department_id
FROM employees
WHERE department_id IN (20, 30, 60);
```

- To select employees who belong to a department other than 20, 30 or 60

```
SELECT employee_id, first_name, last_name, department_id
FROM employees
WHERE department_id NOT IN (20, 30, 60);
```

WHERE clause (contd..) – IS NULL operator

- If a column contains NULL value, the row will not be retrieved by conditional operators.
- For example, if a row has manager_id as NULL, then that row will not be retrieved by where criteria manager_id <> 100
- IS NULL operator retrieve the rows for having NULL in the criteria column

```
SELECT employee_id, first_name, last_name, manager_id
FROM employees
WHERE manager_id IS NULL;
```

- To select employees for whom manager id is any value other than NULL

```
SELECT employee_id, first_name, last_name, manager_id
FROM employees
WHERE manager_id IS NOT NULL;
```

WHERE clause (contd..) – LIKE operator

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column
- There are two wildcards often used in conjunction with the LIKE operator
  - % : indicates zero or more characters
  - _ : indicates one character
- To select employees whose first name starts with “Da”

```
SELECT employee_id, first_name, last_name
FROM employees
WHERE first_name LIKE 'Da%';
```

- To select employees whose first name ends with “er”

```
SELECT employee_id, first_name, last_name
FROM employees
WHERE first_name LIKE '%er';
```

WHERE clause (contd..) – LIKE operator

- To select employees whose first name contains letter “V”

```
SELECT employee_id, first_name, last_name
FROM employees
WHERE first_name LIKE '%v%';
```

- To select employees whose first name has ‘e’ as the second letter

```
SELECT employee_id, first_name, last_name
FROM employees
WHERE first_name LIKE '_e%’;
```

SET Operators

SET Operators

The SQL Set operation is used to combine two or more SQL SELECT Statements.

Types of Set Operation

- UNION
- UNION ALL

Set Operator - UNION

- The SQL UNION operation is used to combine the result of two or more SQL SELECT queries.
- In the UNION operation, all the number of columns and datatype must be same in the SELECT queries on which UNION operation is being applied.
- The UNION operation eliminates the duplicate rows from its result set.

  - **SYNTAX**

```
SELECT column_name FROM table1
UNION
SELECT column_name FROM table2;
```

Set Operator – UNION (contd..)

- HR team want to setup a meeting all employees in UK and all managers in the company. Prepare a list of invitees

```
SELECT employee_id, first_name, last_name, email
FROM employees e
JOIN jobs j ON e.job_id = j.job_id
WHERE j.job_title LIKE '%MANAGER%'
UNION
(SELECT employee_id, first_name, last_name, email
FROM employees e
JOIN departments d ON e.department_id = d.department_id
JOIN locations l ON d.location_id = l.location_id
WHERE l.country_id = 'UK' )
ORDER BY employee_id;
```

Set Operator – UNION ALL

- The UNION ALL operation works similar to the UNION operator.
- The difference in these two operators related to duplicate rows in result set. The UNION ALL operator does not eliminate the duplicate rows while the UNION operator eliminates the duplicate rows from result set.

  - **Syntax**

```
SELECT column_name FROM table1
UNION ALL
SELECT column_name FROM table2;
```

SQL Functions

FUNCTIONS IN SQL

Required to transform the retrieved column values

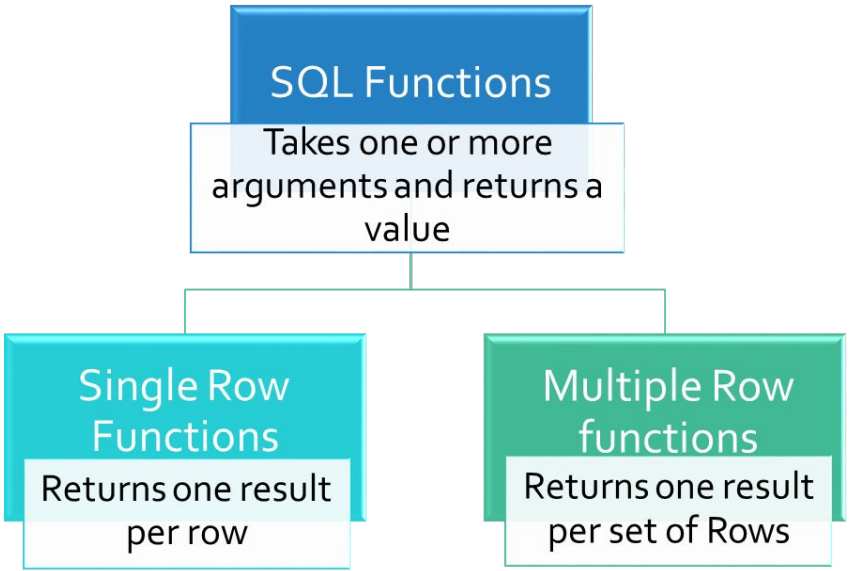
For e.g. rounding off numbers, upper / lower case, date formatting, etc.

Used in expressions, calculations in SQL statements

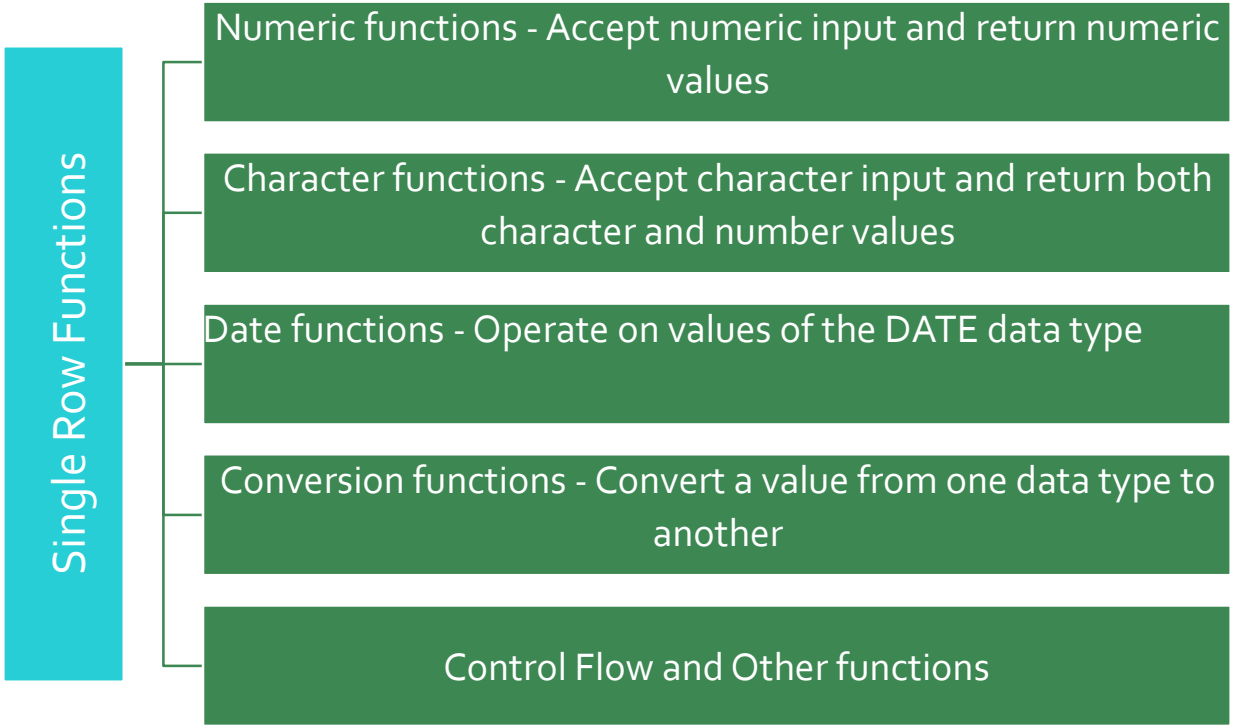
Two categories:

Single-row function: Applied on columns that are passed as parameters to it

Multi-row or Aggregate function: Applied on a group or set of rows



TYPES OF SINGLE ROW FUNCTIONS

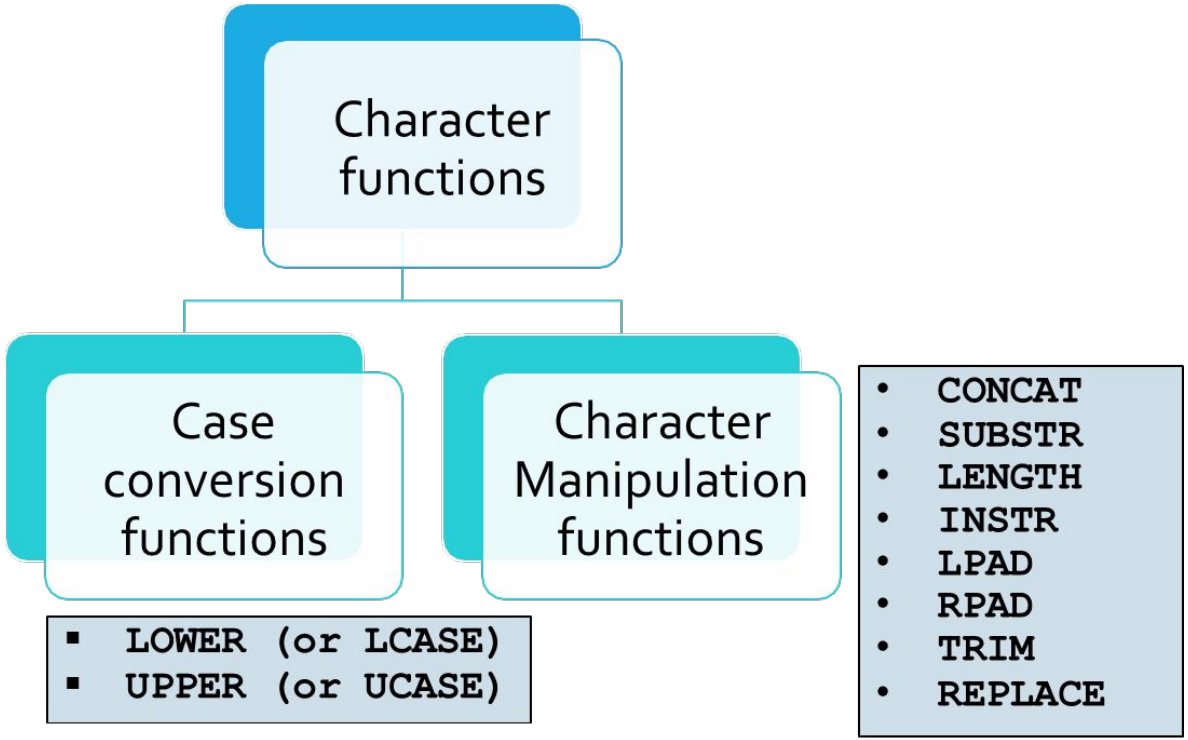


NUMERIC FUNCTIONS

- ROUND: Rounds a numeric value to a specified decimal
- TRUNC: Truncates a numeric value to a specified decimal
- FLOOR: Nearest integer less than the given number
- CEIL: Nearest integer more than the given number

```
SELECT ROUND(45.923,2), ROUND(45.923,0),  ROUND(45.923,-1);

SELECT TRUNC(45.923,2), TRUNC(45.923),  TRUNC(45.923,-1);

SELECT last_name, salary, MOD(salary, 5000)

FROM Employees

WHERE job_id ='SA_REP';
```

TYPES OF CHARACTER FUNCTIONS



CASE CONVERSION FUNCTIONS

Though MySQL string search is case insensitive ('abc' = 'Abc' = 'ABC'), it is advised to keep the case of strings same in both sides while comparing

To make it case insensitive search:

```
SELECT employee_id, last_name, department_id
FROM Employees
WHERE LOWER(last_name) = 'higgins';
```

Above condition can also be written as:

```
WHERE UPPER(last_name) = 'HIGGINS';
```

CHARACTER MANIPULATION FUNCTIONS

Function	Result	Explanation
`CONCAT('Hello', 'World')`	`HelloWorld`	Joins values together (only use two parameters with CONCAT.)
`SUBSTR('HelloWorld',1,5)`	`Hello`	Extracts a string of determined length
`LENGTH('HelloWorld')`	`10`	Shows the length of a string as a numeric value
`INSTR('HelloWorld', 'W')`	`6`	Finds the numeric position of a named character
`LPAD(salary,10,'*')`	`*****24000`	Returns an expression left-padded to the length of *n* characters with a character expression
`RPAD(salary, 10, '*')`	`24000*****`	Returns an expression right-padded to the length of *n* characters with a character expression
`REPLACE ('JACK and JUE','J','BL')`	`BLACK and BLUE`	Returns an expression right-padded to the length of *n* characters with a character expression
`TRIM('H' FROM 'HelloWorld')`	`elloWorld`	Trims leading or trailing characters (or both) from a character string

CHARACTER MANIPULATION FUNCTIONS

One query can contain multiple single-row functions in it:

```
SELECT employee_id,
CONCAT(first_name, ' ', last_name) NAME, job_id,  LENGTH
(last_name),
INSTR(last_name, 'a') "Contains 'a'?"
FROM Employees
WHERE SUBSTR(joob_id,4)='REP'
AND INSTR(last_name, 'a') > 0;
```

WORKING WITH DATES

Date values should be enclosed in single-quotes while using them in a query:

```
SELECT last_name, hire_date
FROM Employees
WHERE hire_date<'1988-02-01'
```

Database stores dates as numbers, it performs calculations using arithmetic operators such as addition and subtraction.
Query to display the no. of weeks the employee has been employed for all employees in department 90

```
SELECT last_name, hire_date, DATEDIFF(CURDATE(),
hire_date)/7 AS Weeks
FROM Employees
WHERE department_id=90
```

DATE FUNCTIONS

Function	Result

`CURRENT_DATE()(or CURDATE())`
`CURRENT_TIME() (or CURTIME())`
`CURRENT_TIMESTAMP() (or NOW())`
	Return current date / time / timestamp

`PERIOD_DIFF(yymm1, yymm2)`	Finds the number of months between two periods.
`ADDDATE(date, n) or DATE_ADD` `SUBDATE(date, n) or DATE_SUB`	Adds / subtracts +/-`n` number of days or interval specified to/from `date`.
`LAST_DAY(date)`	Finds the date of the last day of the month that contains `date`

DATE FUNCTIONS

```
Examples:

SELECT CURRENT_DATE(), CURRENT_TIME();

SELECT PERIOD_DIFF(201810, 201801);

SELECT employee_id, first_name, hire_date,
PERIOD_DIFF(DATE_FORMAT(curdate(), '%Y%m'),
DATE_FORMAT(hire_date, '%Y%m'))/12 as yrs_of_service  FROM
Employees WHERE department_id = 90;

SELECT hire_date, ADDDATE(hire_date, 100) AS  After 100days,
SUBDATE(hire_date, INTERVAL 1 MONTH) AS  Before 1Month FROM
Employees WHERE department_id = 90;

SELECT LAST_DAY(CURDATE());
```

CONVERSION FUNCTIONS

- Data in one form may have to converted to another form before comparing them
- For example, a number '1000' in character format will have to converted before it can be compared with another number
- Some of them are implicitly converted and some of them have to explicitly converted
- MySQL automatically or implicitly converts below data types provided they have valid data
- For e.g. a character 'A' cannot be converted to equivalent number, nor a character-based date, '99-99-9999' can be converted to equivalent date

From	To
VARCHAR or CHAR	NUMBER
VARCHAR or CHAR	DATE
NUMBER	VARCHAR or CHAR
DATE	VARCHAR or CHAR

DATE_FORMAT: DATE TO CHAR

DATE_FORMAT converts a datetime data type to a value of VARCHAR data type in the format specified by the format_specifier

A format specifier is a character literal that describes the format of datetime stored in a character string

Specifier	Description
%a	Abbreviated weekday name (Sun..Sat)
%b	Abbreviated month name (Jan..Dec)
%c	Month, numeric (0..12)
%D	Day of the month with English suffix (0th, 1st, 2nd, 3rd, ...)
%d	Day of the month, numeric (00..31)
%H or %h	Hour (00..23) or (01 .. 12)
%i	Minutes, numeric (00..59)
%j	Day of year (001..366)
%k or %l	Hour (0..23) or (1..12)

Specifier	Description
%M	Month name (January..December)
%m	Month, numeric (00..12)
%p	AM or PM
%r	Time, 12-hour (hh:mm:ss followed by AM or PM)
%S or %s	Seconds (00..59)
%T	Time, 24-hour (hh:mm:ss)
%U or %u	Week (00..53), where Sunday is the first day of the week; WEEK() mode 0
%V or %v	Week (01..53), where Sunday is the first day of the week; WEEK() mode 2; used with %X
%W	Weekday name (Sunday..Saturday)

DATE_FORMAT: DATE TO CHAR

Examples:

```
SELECT DATE_FORMAT(curdate(),'%d-%m-%Y');
SELECT CONCAT(first_name, ' joined on ',
DATE_FORMAT(hire_date, '%W, %M %D, %Y')) AS emp_desc
FROM Employees WHERE employee_id = 123;
SELECT CONCAT('Today is ', DATE_FORMAT(curdate(),  '%W, %D
of %M, %Y')) AS Today;
SELECT CONCAT('Now it''s ', DATE_FORMAT(now(),  '%h:%i:%s
%p'));
```

OTHER DATA TYPE CONVERSION METHODS

Numbers are implicitly converted to CHAR in MySQL.
E.g. SELECT CONCAT('Olympics-', 2019);
Any of the following functions can be used for explicit conversions:
CAST()
CONVERT()

Examples:

```
SELECT CAST('2018-10-31' AS DATE);
SELECT CONVERT('2018-10-31', DATE);

SELECT CAST(150 AS CHAR);
SELECT CONVERT(150, CHAR);

SELECT CAST('15:06:10' AS TIME);
SELECT CONVERT('15:06:10', TIME);
```

CONTROL FLOW FUNCTIONS

Functions that control the flow of the query based on data value:

IF(expr1, expr2, expr3) - If expr1 is True, return expr2, otherwise expr3

IFNULL(expr1, expr2) – If expr1 is not NULL, return expr1, otherwise expr2

NULLIF(expr1, expr2) – If expr1 = expr2, return NULL, otherwise expr1

CASE value WHEN condition THEN result WHEN condition THEN result
ELSE result END - To simulate IF-ELSE ladder in SQL

CONTROL FLOW FUNCTIONS

```
SELECT IF(1=2, 'Y', 'N'); -- Returns 'N'
SELECT salary, IF(salary > 10000, 'High', 'Low') AS sal_grade
FROM Employees WHERE employee_id = 123;

SELECT IFNULL(NULL, 100); -- Returns 100
SELECT employee_id, first_name, department_id, salary,
IFNULL(commission_pct, 0)
FROM Employees WHERE department_id IN (80, 90) AND salary >
10000;

SELECT NULLIF(10, 20); -- Returns 10
SELECT first_name, last_name FROM Employees WHERE
NULLIF(LEFT(first_name, 1), LEFT(last_name, 1)) IS NULL;
```

Analytic Functions

ANALYTIC FUNCTIONS

Analytic functions calculate an aggregate value based on a group of rows

However unlike aggregate functions, analytic functions return multiple rows for each group.

Can be used to compute moving averages, running totals, percentages or top-N results within a group

Some examples: CORR, CUME_DIST, LAG, LEAD, LISTAGG, NTH_VALUE, RANK, DENSE_RANK, etc.

Aggregate functions like AVG, SUM, etc. also can be used as analytic functions

Syntax:

```
Select …from table analytic_function([ arguments ])
OVER (( [PARTITION BY <...>] [ORDER BY <....>]
[<window_clause>] )
```

ANALYTIC FUNCTIONS: BASIC AGGREGATE FUNCTIONS

- COUNT

```
SELECT last_name, department_id,
COUNT(*) OVER (PARTITION BY department_id) dept_cnt  FROM
Employees
WHERE department_id IN (30, 40);
```

- SUM

```
SELECT employee_id, last_name, SUM(salary)
OVER (PARTITION BY department_id) dept_total, department_id  FROM
Employees
WHERE department_id IN (30, 40);
```

- AVG

```
SELECT employee_id, last_name, AVG(salary)
OVER (PARTITION BY department_id) dept_Avg,department_id  FROM
Employees
WHERE department_id IN (30, 40);
```

ANALYTIC FUNCTIONS

- ROW_NUMBER():- Gives a running serial number to a partition of records
- RANK():- Calculates the rank of a value in a group of values. The return type is NUMBER.
- DENSE_RANK():- Acts like the RANK function except that it assigns consecutive ranks
- LEAD computes an expression based on the next rows. i.e. rows coming after the current row and return value to current row
    - LEAD (expr, offset, default)
      expr = expression to compute from leading row
      offset = index of the leading row relative to the current row
      default = value to return if the <offset> points to a row beyond partition range
- FIRST_VALUE returns the first result from an ordered set

ANALYTIC FUNCTIONS

- CUME_DIST - display the cumulative distribution, or the relative position in the set, of each of the employees, as well as all the original data.

- NTILE - Breaks a result set into a specified number of approximately equal groups, or buckets, rows permitting. If the no. of rows in the set is smaller than the number of buckets specified, the number of buckets will be reduced so there is one row per bucket.

- PERCENT_RANK - Assigns value between 0-1 which represents the position of the current row relative to the set as a percentage

SUMMARY

- Single row functions accept one or more arguments and return one result per row.
- Conversion functions are used to convert one data type to another with a specified format.
- Group functions accept one or more arguments and return one result for multiple rows
- Different types of joins are used for retrieving data from multiple tables
- Set operations on tables are UNION and INTERSECT
- Analytic functions or windowing functions work similar to group functions
- Analytic functions do not reduce the no. of rows in the result unlike group functions

Dimension Functions

DIMENSION FUNCTIONS

In data warehouse, a dimension is a collection of reference information about a measurable event

The events are referred as facts and are stored in fact tables

For example, dimension tables store data like Customer details, Product details, Sales information, etc.

Fact tables contain the foreign key to each of these dimension tables with the corresponding measure say, total sales, average sales, spend, etc.

Functions like rollup, cube, etc. are required in the data warehouse context to aggregate the data in the fact table levels

ROLLUP

A simple extension to the GROUP BY clause.

Enables a SELECT statement to calculate multiple levels of subtotals across a specified group of dimensions.
Also calculates a grand total.

Highly efficient, adding minimal overhead to a query, easy to use.

```
SELECT department_id, SUM(salary)
FROM Employees
WHERE department_id IS NOT NULL  GROUP BY department_id
WITH ROLLUP;
SELECT department_id,job_id, sum(salary)  FROM Employees
WHERE department_id IS NOT NULL
GROUP BY department_id, job_id WITH ROLLUP;
```

GROUPING SETS

GROUPING SETS are a further extension of the GROUP BY clause .
It lets you specify multiple groupings of data
The output of ROLLUP includes the rows produced by the regular GROUP BY operation along with the summary rows
GROUPING SETS used to generate summary information at the level you choose without including all the rows produced by the regular GROUP BY operation.

```
SELECT department_id,job_id, SUM(salary),
GROUPING(department_id), GROUPING(job_id)
FROM Employees
GROUP BY department_id, job_id WITH ROLLUP;
```

IF & CASE Statement

IFNULL Statement

- IFNULL statement evaluates the first expression. If it is not NULL then it will return the value of the expression. Else it will return the alternate value

- Syntax

  IFNULL(expression, alternate_value)

- While displaying employee information, set department to "To Be Assigned" if the department id is NULL

```
SELECT employee_id, last_name,
       IFNULL(department_id, "To Be Assigned") AS department
FROM   employees
WHERE  employee_id > 170;
```

IF Statement

- IF statement takes three parameters: condition, expression_if_true, expression_if_false.
- It evaluates a condition. If the condition is True, then it returns the the second parameter (expression_if_true), else returns the third parameter (expression_if_false)

- Syntax

```
IF(condition, expression_if_true, expression_if_false);
```

IF Statement (contd..)

- To display eligibility of employees based on hire date

```
SELECT employee_id, last_name,      hire_date,
   IF(hire_date <= '1999-12-31'    , "Eligible",     "Not Eligible") AS
scheme_eligibility
FROM employees;
```

- Based on designation display whether employee is manager or not

```
SELECT employee_id, last_name,
    IF(job_id    LIKE '%MGR%' OR job_id LIKE   "%MAN%", "Yes",'No') AS  Manager
FROM  employees;
```

CASE Statement

- The CASE statement goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

- If there is no ELSE part and no conditions are true, it returns NULL.

- Syntax

```
CASE
   WHEN condition1 THEN expression1
   WHEN condition2 THEN expression2....
   WHEN conditionN THEN expressionn
   ELSE expression
END;
```

CASE Statement (contd..)

- To display appropriate tax slab based on salary

```
SELECT employee_id, last_name, salary,
      CASE
         WHEN  salary <=  5000 THEN  'Tax slab  A'
         WHEN salary <= 10000 THEN 'Tax slab  B'
         WHEN salary <= 15000 THEN 'Tax slab  C'
         ELSE  'Tax slab D'
      END  AS  taxation
FROM employees;
```

- To display proposed salary based on job id

```
SELECT employee_id, last_name, job_id, salary,
   CASE
      WHEN  job_id = 'SA_REP' THENsalary    * 1.25
      WHEN  job_id = 'IT_PROG' THEN salary    * 1.2
      ELSE  salary * 1.05
   END  AS proposed_salary
FROM  employees;
```

ORDERBY Clause

SELECT	Specifies columns to be displayed in the result
FROM	Specifies tables from which data will be queried
WHERE	Specifies criteria for filtering records (Optional)
GROUP BY	Specifies columns used for grouping the data (Optional)
HAVING	Specifies criteria for filtering grouped data (Optional)
ORDER BY	Specifies sort order in which result is displayed (Optional)

ORDER BY Clause

- The ORDER BY clause sorts the result-set in asce nding or descending order.
- By default, it sorts the records in ascending order. DESC keyword is used to sort the records in descending order
- ORDER BY is the last clause in the SELECT statement
- Column alias can used in the ORDER BY clause
- In MySQL, Null values are displayed at first for ascending sequences and at last for descending sequences

- **SYNTAX**

```
SELECT col1,col2
FROM table_name
WHERE condition
ORDER BY col1, col2 DESC;
```

ORDER BY clause (contd..)

- To sort employees in the Ascending order of employee id

```
SELECT employee_id, first_name, last_name
FROM  employees
ORDER BY employee_id;
```

- To sort employees in the Descending order of hire date

```
SELE CT employee_id, first_name , last_name, hire_date
FROM emp loyees
ORDER BY hire_date DESC ;
```

ORDER BY clause (contd..)

- To sort employees in the Ascending order of department id and by Descending order of salary within department id

```
SELECT first_name, last_name, department_id, hire_date
FROM employees
ORDER BY department_id, hire_date DESC;
```

- Note – in the above statement DESC is applicable for hire_date. Department id is sorted in Ascending order (since ASC is default)
- The above statement can also be written as follows. The numbers 3 and 4 in the ORDER BY clause indicate the position of column in the SELECT clause

```
SELECT first_name,      last_name,     department_id, salary
FROM employees
ORDER BY 3, 4 DESC;
```

ORDER BY clause (contd..) – LIMIT clause

- LIMIT clause is used for limiting the number rows retrieved to a specified number
- LIMIT clause combined with ORDER BY clause is useful to retrieve the 'first few' rows, e.g., retrieve 10 earliest joined employees, 5 highest paid employees etc.
- To select the 10 earliest joined employees in department 80

```
SELECT *
FROM employees
WHERE department_id        = 80
ORDER BY hire_date
LIMIT 10;
```