1. What do you mean by single row functions? List the different types of single row functions.

1. Single Row Functions

Single row functions (or scalar functions) are functions that operate on a single row of data at a time and return one result for each row processed. If you apply a single row function to a column in a table with ten rows, you will get ten results back.

The different types of single row functions are:

- **Numeric Functions**: Perform operations on numbers (e.g., rounding, absolute value).
- **String (or Character) Functions**: Manipulate text strings (e.g., changing case, concatenating).
- **Date Functions**: Perform operations on date and time values.
- **Conversion Functions**: Convert a value from one data type to another (e.g., string to number).
- **General Functions**: Handle null values or provide conditional logic (e.g., `NVL`, `COALESCE`, `DECODE`, `CASE`).

2. Explain the following numeric functions with examples: Abs, Sign, Sqrt, Mod, Power, Exp, Ln, Log, Ceil, Floor, Round, Trunc, Greatest, Least.

2. Numeric Functions 🔢

Numeric functions accept numeric input and return numeric values. The examples below use the `DUAL` table, a special one-row, one-column table present in some databases like Oracle, which is useful for demonstrating functions.

Abs

- **Purpose**: Returns the absolute (non-negative) value of a number.
- **Syntax**: `ABS(number)`
- **Example**:

  SQL

  ```
  SELECT ABS(-15.5) FROM DUAL;
  -- Result: 15.5
  ```

Sign

- **Purpose**: Returns the sign of a number: -1 for negative, 0 for zero, and 1 for positive.
- **Syntax**: `SIGN(number)`
- **Example**:

  SQL

  ```
  SELECT SIGN(-250), SIGN(0), SIGN(250) FROM DUAL;
  -- Result: -1, 0, 1
  ```

Sqrt

- **Purpose**: Returns the square root of a non-negative number.
- **Syntax**: `SQRT(number)`
- **Example**:

  SQL

  ```
  SELECT SQRT(81) FROM DUAL;
  -- Result: 9
  ```

Mod

- **Purpose**: Returns the remainder of a division operation.
- **Syntax**: `MOD(dividend, divisor)`
- **Example**:

  SQL

  ```
  SELECT MOD(10, 3) FROM DUAL;
  -- Result: 1 (because 10 / 3 = 3 with a remainder of 1)
  ```

Power

- **Purpose**: Returns a number raised to the power of another number.
- **Syntax**: `POWER(base, exponent)`
- **Example**:

  SQL

  ```
  SELECT POWER(3, 4) FROM DUAL;
  -- Result: 81 (because 3 * 3 * 3 * 3 = 81)
  ```

Exp

- **Purpose**: Returns the value of e (Euler's number, approx. 2.71828) raised to the power of a given number.
- **Syntax**: `EXP(number)`
- **Example**:

  SQL

  ```
  SELECT EXP(1) FROM DUAL;
  -- Result: 2.71828...
  ```

Ln

- **Purpose**: Returns the natural logarithm (base e) of a number.
- **Syntax**: `LN(number)`
- **Example**:

  SQL

  ```
  SELECT LN(2.71828) FROM DUAL;
  -- Result: ~1
  ```

Log

- **Purpose**: Returns the logarithm of a number to a specified base.
- **Syntax**: `LOG(base, number)`
- **Example**:

  SQL

  ```
  SELECT LOG(10, 100) FROM DUAL;
  -- Result: 2 (because 10^2 = 100)
  ```

Ceil

- **Purpose**: Rounds a number **up** to the nearest integer.
- **Syntax**: `CEIL(number)`
- **Example**:

  SQL

  ```
  SELECT CEIL(9.01) FROM DUAL;
  -- Result: 10
  ```

Floor

- **Purpose**: Rounds a number **down** to the nearest integer.
- **Syntax**: `FLOOR(number)`
- **Example**:

  SQL

  ```
  SELECT FLOOR(9.99) FROM DUAL;
  -- Result: 9
  ```

Round

- **Purpose**: Rounds a number to a specified number of decimal places.
- **Syntax**: `ROUND(number, decimal_places)`
- **Example**:

  SQL

  ```
  SELECT ROUND(123.456, 2), ROUND(123.456, 0) FROM DUAL;
  -- Result: 123.46, 123
  ```

Trunc

- **Purpose**: Truncates (cuts off) a number to a specified number of decimal places without rounding.
- **Syntax**: `TRUNC(number, decimal_places)`
- **Example**:

  SQL

  ```
  SELECT TRUNC(123.456, 2), TRUNC(123.456, 0) FROM DUAL;
  -- Result: 123.45, 123
  ```

Greatest

- **Purpose**: Returns the largest value from a list of expressions.
- **Syntax**: `GREATEST(value1, value2, ...)`
- **Example**:

  SQL

  ```
  SELECT GREATEST(5, 18, 2, 11) FROM DUAL;
  -- Result: 18
  ```

Least

- **Purpose**: Returns the smallest value from a list of expressions.
- **Syntax**: `LEAST(value1, value2, ...)`
- **Example**:

  SQL

  ```
  SELECT LEAST(5, 18, 2, 11) FROM DUAL;
  -- Result: 2
  ```

3. Explain the following string functions with examples: Initcap, Upper, Lower, Length, Rpad, Lpad, Ltrim, Rtrim, Trim, Translate, Replace, Concat, Ascii, Chr, Substr, Instr, Greatest, Least.

3. String Functions 📝

String (or character) functions accept string input and can return both character and numeric values.

Initcap

- **Purpose**: Converts the first letter of each word to uppercase and all other letters to lowercase.
- **Syntax**: `INITCAP(string)`
- **Example**:

  SQL

  ```
  SELECT INITCAP('hello world') FROM DUAL;
  -- Result: 'Hello World'
  ```

Upper

- **Purpose**: Converts all letters in a string to uppercase.
- **Syntax**: `UPPER(string)`
- **Example**:

  SQL

  ```
  SELECT UPPER('gemini') FROM DUAL;
  -- Result: 'GEMINI'
  ```

Lower

- **Purpose**: Converts all letters in a string to lowercase.
- **Syntax**: `LOWER(string)`
- **Example**:

  SQL

  ```
  SELECT LOWER('GEMINI') FROM DUAL;
  -- Result: 'gemini'
  ```

Length

- **Purpose**: Returns the number of characters in a string.
- **Syntax**: `LENGTH(string)`
- **Example**:

  SQL

  ```
  SELECT LENGTH('Gemini') FROM DUAL;
  -- Result: 6
  ```

Rpad / Lpad

- **Purpose**: Pads a string on the right (`RPAD`) or left (`LPAD`) with a specified character to a certain length.
- **Syntax**: `RPAD(string, length, pad_char)`,`LPAD(string, length, pad_char)`
- **Example**:

  SQL

  ```
  SELECT RPAD('SQL', 7, '*'), LPAD('SQL', 7, '-') FROM DUAL;
  -- Result: 'SQL****', '----SQL'
  ```

Rtrim / Ltrim

- **Purpose**: Removes characters from the right (`RTRIM`) or left (`LTRIM`) of a string. By default, it removes spaces.
- **Syntax**: `RTRIM(string, [trim_chars])`,`LTRIM(string, [trim_chars])`
- **Example**:

  SQL

  ```
  SELECT LTRIM('  Hello'), RTRIM('Hello..', '.') FROM DUAL;
  -- Result: 'Hello', 'Hello'
  ```

Trim

- **Purpose**: Removes leading, trailing, or both kinds of characters from a string.
- **Syntax**: `TRIM([LEADING|TRAILING|BOTH] [trim_char FROM] string)`
- **Example**:

  SQL

  ```
  SELECT TRIM(' ' FROM '  Hello World  ') FROM DUAL;
  -- Result: 'Hello World'
  ```

Translate

- **Purpose**: Replaces each character in a string with another character on a one-to-one basis.
- **Syntax**: `TRANSLATE(string, from_string, to_string)`
- **Example**:

  SQL

  ```
  SELECT TRANSLATE('12345', '15', 'AB') FROM DUAL;
  -- Result: 'A234B' (1 is replaced by A, 5 is replaced by B)
  ```

Replace

- **Purpose**: Replaces every occurrence of a sequence of characters with another sequence.
- **Syntax**: `REPLACE(string, search_string, replace_string)`
- **Example**:

  SQL

  ```
  SELECT REPLACE('Jack and Jill', 'J', 'Bl') FROM DUAL;
  -- Result: 'Black and Bill'
  ```

Concat

- **Purpose**: Joins two strings together. The `||` operator is often used for the same purpose.
- **Syntax**: `CONCAT(string1, string2)`
- **Example**:

  SQL

  ```
  SELECT CONCAT('Hello', ' World') FROM DUAL;
  -- Result: 'Hello World'
  ```

Ascii / Chr

- **Purpose**: `ASCII` returns the numeric ASCII code for the first character of a string. `CHR` returns the character for a given ASCII code.
- **Syntax**: `ASCII(string)`, `CHR(number)`
- **Example**:

  SQL

  ```
  SELECT ASCII('A'), CHR(65) FROM DUAL;
  -- Result: 65, 'A'
  ```

Substr

- **Purpose**: Extracts a substring of a specified length starting from a given position.
- **Syntax**: `SUBSTR(string, start_position, length)`
- **Example**:

  SQL

  ```
  SELECT SUBSTR('Hello World', 7, 5) FROM DUAL;
  -- Result: 'World'
  ```

Instr

- **Purpose**: Finds the starting position of a substring within a string.
- **Syntax**: `INSTR(string, substring, [start_position], [occurrence])`
- **Example**:

  SQL

  ```
  SELECT INSTR('corporate', 'or', 1, 2) FROM DUAL;
  -- Result: 5 (finds the 2nd occurrence of 'or' starting from position
  1)
  ```

Greatest / Least

- **Purpose**: Returns the lexicographically largest (`GREATEST`) or smallest (`LEAST`) string from a list.
- **Syntax**: `GREATEST(string1, ...)`, `LEAST(string1, ...)`
- **Example**:

  SQL

  ```
  SELECT GREATEST('Apple', 'Zebra', 'Banana'), LEAST('Apple', 'Zebra',
  'Banana') FROM DUAL;
  -- Result: 'Zebra', 'Apple'
  ```

4. Explain the following date function with examples: Sysdate, Current_date, Current_timestamp, Systimestamp, To_Char, To_Date, Add_months, Months_between, Next_day, Last_day, Greatest, Least.

4. Date Functions 📅

Date functions operate on values of the `DATE` data type.

Sysdate / Current_date

- **Purpose**: `SYSDATE` returns the current date and time from the database server. `CURRENT_DATE` returns the current date in the user's session time zone.
- **Syntax**: `SYSDATE`, `CURRENT_DATE`
- **Example**:

  SQL

  ```
  SELECT SYSDATE, CURRENT_DATE FROM DUAL;
  -- Result: 08-SEP-25, 08-SEP-25 (format may vary)
  ```

Systimestamp / Current_timestamp

- **Purpose**: Similar to the above but with higher precision (fractional seconds) and time zone information.
- **Syntax**: `SYSTIMESTAMP`, `CURRENT_TIMESTAMP`
- **Example**:

  SQL

  ```
  SELECT SYSTIMESTAMP FROM DUAL;
  -- Result: 08-SEP-25 02.10.11.123456 PM +05:30
  ```

To_Char

- **Purpose**: Converts a `DATE` or `NUMBER` value to a string, using a specified format.
- **Syntax**: `TO_CHAR(date, 'format_model')`
- **Example**:

  SQL

  ```
  SELECT TO_CHAR(SYSDATE, 'Day, DD Month YYYY HH:MI:SS AM') FROM DUAL;
  -- Result: 'Monday   , 08 September 2025 02:10:11 PM'
  ```

To_Date

- **Purpose**: Converts a string containing a date into a `DATE` data type, based on a specified format.
- **Syntax**: `TO_DATE(string, 'format_model')`
- **Example**:

  SQL

  ```
  SELECT TO_DATE('July 4, 2025', 'Month DD, YYYY') FROM DUAL;
  -- Result: A date value for 04-JUL-25
  ```

Add_months

- **Purpose**: Adds a specified number of months to a date.
- **Syntax**: `ADD_MONTHS(date, number_of_months)`
- **Example**:

  SQL

  ```
  SELECT ADD_MONTHS(TO_DATE('15-Jan-2025', 'DD-Mon-YYYY'), 3) FROM
  DUAL;
  -- Result: A date value for 15-APR-25
  ```

Months_between

- **Purpose**: Returns the number of months between two dates.
- **Syntax**: `MONTHS_BETWEEN(date1, date2)`
- **Example**:

  SQL

  ```
  SELECT MONTHS_BETWEEN('01-JAN-2026', '01-OCT-2025') FROM DUAL;
  -- Result: 3
  ```

Next_day

- **Purpose**: Returns the date of the first specified day of the week that is later than the given date.
- **Syntax**: `NEXT_DAY(date, 'day_of_week')`
- **Example**:

  SQL

  ```
  -- If today is 08-Sep-2025 (Monday)
  SELECT NEXT_DAY(SYSDATE, 'FRIDAY') FROM DUAL;
  -- Result: A date value for 12-SEP-25
  ```

Last_day

- **Purpose**: Returns the date of the last day of the month that contains the specified date.
- **Syntax**: `LAST_DAY(date)`
- **Example**:

  SQL

  ```
  SELECT LAST_DAY(TO_DATE('10-FEB-2024', 'DD-MON-YYYY')) FROM DUAL;
  -- Result: A date value for 29-FEB-24 (2024 is a leap year)
  ```

Greatest / Least

- **Purpose**: Returns the latest (`GREATEST`) or earliest (`LEAST`) date from a list.
- **Syntax**: `GREATEST(date1, ...)` , `LEAST(date1, ...)`
- **Example**:

  SQL

  ```
  SELECT GREATEST(TO_DATE('01-Jan-25'), TO_DATE('01-Mar-25'),
  TO_DATE('01-Feb-25')) FROM DUAL;
  -- Result: A date value for 01-MAR-25
  ```

5. What do you mean by Group functions? Explaint he following group functions with example: Sum, Avg, Max, Min, Count.

5. Group Functions 📊

Group functions (or aggregate functions) operate on a set of rows to return a single summary result. They are commonly used with the `GROUP BY` clause to calculate metrics for subgroups of data.

For the examples, assume we have an `employees` table:

emp_id	dept_id	salary
101	10	5000
102	20	6000
103	10	7000
104	20	8000
105	10	null

Sum

- **Purpose**: Calculates the sum of all values in a numeric column. It ignores `NULL` values.
- **Syntax**: `SUM(expression)`
- **Example**:

  SQL

  ```
  -- Sum of all salaries
  SELECT SUM(salary) FROM employees;
  -- Result: 26000

  -- Sum of salaries for each department
  SELECT dept_id, SUM(salary) FROM employees GROUP BY dept_id;
  -- Result:
  -- 10, 12000
  -- 20, 14000
  ```

Avg

- **Purpose**: Calculates the average of all values in a numeric column. It ignores `NULL` values.
- **Syntax**: `AVG(expression)`
- **Example**:

  SQL

  ```
  -- Average of all salaries
  SELECT AVG(salary) FROM employees;
  -- Result: 6500 (26000 / 4)

  -- Average salary for each department
  SELECT dept_id, AVG(salary) FROM employees GROUP BY dept_id;
  -- Result:
  -- 10, 6000
  -- 20, 7000
  ```

Max

- **Purpose**: Finds the maximum value in a column.
- **Syntax**: `MAX(expression)`
- **Example**:

  SQL

  ```
  -- Highest salary in the company
  SELECT MAX(salary) FROM employees;
  -- Result: 8000

  -- Highest salary in each department
  SELECT dept_id, MAX(salary) FROM employees GROUP BY dept_id;
  -- Result:
  -- 10, 7000
  -- 20, 8000
  ```

Min

- **Purpose**: Finds the minimum value in a column.
- **Syntax**: `MIN(expression)`
- **Example**:

  SQL

  ```
  -- Lowest salary in the company
  SELECT MIN(salary) FROM employees;
  -- Result: 5000

  -- Lowest salary in each department
  SELECT dept_id, MIN(salary) FROM employees GROUP BY dept_id;
  -- Result:
  -- 10, 5000
  -- 20, 6000
  ```

Count

- **Purpose**: Counts the number of rows. `COUNT(*)` counts all rows, while `COUNT(column)` counts non-`NULL` values in that column.
- **Syntax**: `COUNT(* | expression)`
- **Example**:

  SQL

  ```
  -- Total number of employees
  SELECT COUNT(*) FROM employees;
  -- Result: 5

  -- Number of employees with a non-null salary
  SELECT COUNT(salary) FROM employees;
  -- Result: 4

  -- Number of employees in each department
  SELECT dept_id, COUNT(*) FROM employees GROUP BY dept_id;
  -- Result:
  -- 10, 3
  -- 20, 2
  ```