Data Engineering Batch 1 **Day 5: 23/01/2024**

Name: Pradip Bochare

Statement: Data cleaning & Transformation queries, Ranking function, stored procedure

- Data Cleaning and Transformation queries

❖ Handling Missing Values:

- Identifying Missing Values: Use a SELECT statement to find rows where a specific column has NULL values.
  ```
  SELECT  *
  FROM your_table
  WHERE column_name IS NULL;
  ```

- Filling Missing Values: Use an UPDATE statement to replace NULL values in a specific column with a default value.
  ```
  UPDATE your_table
  SET column_name = default_value
  WHERE column_name IS NULL;
  ```

- Dropping Rows with Missing Values: Use a DELETE statement to remove rows where a specific column has NULL values.
  ```
  DELETE FROM your_table
  WHERE column_name IS NULL;
  ```

❖ Handling Duplicates:

- Identifying Duplicates: Use a SELECT statement with GROUP BY and HAVING to find duplicate values in a specific column.
  ```
  SELECT column_name, COUNT(*)
  FROM your_table
  GROUP BY column_name
  HAVING COUNT(*) > 1;
  ```

- Removing Duplicates: Use a DELETE statement with a subquery to keep only the row with the minimum ROWID (or another unique identifier) for each duplicate set.

  ```
  DELETE FROM your_table
  WHERE column_name IN (
  SELECT column_name
   FROM your_table
  GROUP BY column_name
  HAVING COUNT(*) > 1)
  AND ROWID NOT IN (
  SELECT MIN(ROWID)
  FROM your_table
  GROUP BY column_name
  );
  ```

❖ Data Type Conversion:

- Converting Data Types: Use ALTER TABLE to modify the data type of a specific column.

  ```
  ALTER TABLE your_table
  MODIFY column_name new_data_type;
  ```

❖ String Operations:

- Changing Case: Use an UPDATE statement with the UPPER or LOWER function to change the case of values in a specific column.

  ```
  UPDATE your_table
  SET column_name = UPPER(column_name);
  ```

- Trimming Spaces: Use an UPDATE statement with the TRIM function to remove leading and trailing spaces from values in a specific column.

  ```
  UPDATE your_table
  SET column_name = TRIM(column_name);
  ```

- Date Operations:

  - Converting Strings to Dates: Use an UPDATE statement with TO_DATE to convert string representations of dates to the date data type.
    ```
    UPDATE your_table
    SET date_column = TO_DATE(date_string, 'YYYY-MM-DD');
    ```

  - Extracting Components: Use SELECT with EXTRACT to retrieve specific components (e.g., year) from date values.
    ```
    SELECT EXTRACT(YEAR FROM date_column) AS year
    FROM your_table;
    ```

- Aggregations and Grouping:

  - Summarizing Data: Use SELECT with COUNT, AVG, or other aggregate functions along with GROUP BY to summarize data based on a specific column.
    ```
    SELECT column_name, COUNT(*), AVG(some_numeric_column)
    FROM your_table
    GROUP BY column_name;
    ```

- Combining Data:

  - Joining Tables: Use SELECT with INNER JOIN to combine data from two tables based on a common column.
    ```
    SELECT *
    FROM table1
    INNER JOIN table2 ON table1.id = table2.id;
    ```

  - Concatenating Strings: Use SELECT with CONCAT (or the appropriate concatenation function) to combine values from multiple columns into a single string.
    ```
    SELECT CONCAT(column1, ' ', column2) AS concatenated_column
    FROM your_table;
    ```

Ranking Function:
Ranking functions in SQL are used to assign a rank to each row within a result set based on the values in one or more columns. There are several ranking functions available in SQL, and they are often used in analytical queries.

- ROW_NUMBER():
  Assigns a unique number to each row based on the order specified in the ORDER BY clause.

```
SELECT
   column1,
   column2,
   ROW_NUMBER() OVER (ORDER BY column3) AS row_num
FROM your_table;
```

- RANK():
  Assigns a rank to each distinct row within a result set. Rows with the same values get the same rank, and the next rank is skipped.

```
SELECT
   column1,
   column2,
   RANK() OVER (ORDER BY column3) AS ranking
FROM your_table;
```

- DENSE_RANK():
  Similar to RANK(), but without skipping ranks for tied values. It assigns a unique rank to each distinct row, and tied values receive the same rank without gaps.

```
SELECT
   column1,
   column2,
   DENSE_RANK() OVER (ORDER BY column3) AS dense_rank
FROM your_table;
```

- NTILE(n):
  Divides the result set into "n" number of roughly equal parts and assigns a bucket number to each row.

```
SELECT
    column1,
    column2,
    NTILE(4) OVER (ORDER BY column3) AS quartile
FROM your_table;
```

- PERCENT_RANK():
  Calculates the relative rank of a row as a percentage. Useful for comparing a row's position within a result set.

```
SELECT
    column1,
    column2,
    PERCENT_RANK() OVER (ORDER BY column3) AS percent_rank
FROM your_table;
```

- Stored Procedure

  A stored procedure is a precompiled collection of one or more SQL statements or procedural statements, which are stored as a single unit in a database management system. It is a type of database object that allows you to group a set of SQL statements into a reusable and well-defined interface.

  - Creation of Stored Procedures:

    ```
    CREATE PROCEDURE procedure_name
    AS
    BEGIN
       -- SQL statements and procedural code here
    END;
    ```

  - Parameters: Stored procedures can accept input parameters, allowing you to pass values into the procedure when it is called.

    ```
    CREATE PROCEDURE procedure_with_parameters
       @param1 datatype,
       @param2 datatype
    AS
    BEGIN
       -- SQL statements using @param1 and @param2
    END;
    ```

  - Execution: Stored procedures are executed using a CALL or EXEC statement, depending on the database system.

    ```
    EXEC procedure_name; -- SQL Server, MySQL
    ```

- Example Of Stored Procedure

  - Creating Stored Procedure

```
CREATE PROCEDURE GetEmployeeInfo
   @employeeId INT,
   @departmentId INT
AS
BEGIN
   -- Selecting information based on parameters
   SELECT
      EmployeeID,
      FirstName,
      LastName,
      DepartmentID
   FROM
      Employees
   WHERE
      EmployeeID = @employeeId
      AND DepartmentID = @departmentId;
END;
```

  - Executing Stored Procedure

```
EXEC GetEmployeeInfo @employeeId = 101, @departmentId = 1;
```