DATA CLEANING USING SQL

In the rapidly evolving landscape of data-driven decision-making, the foundation of any insightful analysis rests on the quality of the underlying data. Raw data, often heterogeneous and messy, requires meticulous refinement to unveil the valuable insights hidden within. One remarkable endeavor in this realm is the art of data cleaning – a process that goes beyond the mundane task of removing anomalies. It's a transformative journey that involves unraveling complexity, taming inconsistencies, and molding raw information into a coherent and reliable foundation for analysis.

SQL Commands Used in This Project:

Functions:

- MIN | ISNULL | SUBSTRING | REPLACE | CHARINDEX | LEN | PARSENAME

Operators:

- AND | ASC | DESC

Clauses:

- AS | WHERE | GROUP BY | HAVING | ORDER BY | DISTINCT | JOIN

Statements:

- CASE | WHEN | THEN

Data cleaning in SQL involves preparing and transforming raw data in a database to make it suitable for analysis, reporting, or other purposes. Here

are some common data cleaning tasks you might perform using SQL :

1] CONVERTING DATA TYPES :

- A query to alter the data type of the 'SaleDate' column in the 'housing' table from DATETIME to DATE. This type of query is crucial when you need to standardize the data type for a particular column, ensuring that the values match the desired format for better consistency and accuracy in analysis.

```
--1]CONVERTING THE DATA TYPE IN 'SaleDate' COLUMN FROM DATETIME TO DATE

ALTER TABLE housing
ALTER COLUMN SaleDate DATE;
```

2] POPULATING THE NULL VALUE FIELDS :

- This query identifies rows in the 'housing' table where 'PropertyAddress' is NULL, and there are multiple rows with the same 'ParcelID' but different 'UniqueID'. It showcases situations where there might be multiple records for the same property, but only some of them have missing addresses.

```
--2]POPULATING PROPERTY ADDRESS DATA

    --SHOWCASES ALL THE NULL ROWS WHICH HAS SAME 'ParcelID' BUT DIFFERENT 'UniqueID' BETWEEN THE TWO COLUMNS

    SELECT a.ParcelID, a.PropertyAddress, b.ParcelID, b.PropertyAddress
    FROM housing AS a
    join
    housing AS b
    ON
        a.ParcelID = b.ParcelID
        and a.[UniqueID ] <> b.[UniqueID ]
    WHERE
        a.PropertyAddress IS NULL;
```

- This query updates the NULL 'PropertyAddress' values in the 'housing' table by using the non-NULL 'PropertyAddress' value from another row with the same 'ParcelID' but different 'UniqueID'. This way, you're essentially copying the address from another record for the same property.

```
--2]POPULATING PROPERTY ADDRESS DATA

    --UPDATING THE NULL VALUES

    UPDATE a
    SET PropertyAddress = ISNULL(a.PropertyAddress,b.PropertyAddress)
    FROM housing AS a
    join
    housing AS b
    ON
        a.ParcelID = b.ParcelID
        and a.[UniqueID ] <> b.[UniqueID ]
    WHERE
        a.PropertyAddress is null;
```

3] SPLITTING VALUES (PropertyAddress) :

- This query demonstrates how to split the values in the 'PropertyAddress' column into 'Address' and 'City' parts using the SUBSTRING function along with CHARINDEX and LEN.

```
--3]SPLITTING THE VALUES IN 'PropertyAddress' INTO ADDRESS, CITY

    ----SHOWCASING THE SPLIT VALUES
    SELECT
    SUBSTRING (PropertyAddress, 1, CHARINDEX(',', PropertyAddress) -1) AS Address,
    SUBSTRING (PropertyAddress, CHARINDEX(',', PropertyAddress) +1, LEN(PropertyAddress)) AS City
    FROM housing;
```

- These queries involve altering the 'housing' table by adding 'Prop_Address' and 'Prop_City' columns to store the split values. Then, the existing 'PropertyAddress' column is updated to fill the 'Prop_Address' column with the address part and the 'Prop_City' column with the city part.

```
--3]SPLITTING THE VALUES IN 'PropertyAddress' INTO ADDRESS, CITY


    ----ADDING COLUMNS 'Address' & 'City' AND UPDATING THEM WITH THE SPLIT VALUES
    ALTER TABLE housing
    ADD
        Prop_Address nvarchar(255);

    UPDATE housing
    SET
        Prop_Address = SUBSTRING (PropertyAddress, 1, CHARINDEX(',', PropertyAddress) -1);


    ALTER TABLE housing
    ADD
        Prop_City nvarchar(255);

    UPDATE housing
    SET
        Prop_City = SUBSTRING (PropertyAddress, CHARINDEX(',', PropertyAddress) +1, LEN(PropertyAddress));
```

4] SPLITTING ADDRESS (OwnerAddress) :

- This query demonstrates how to split the values in the 'OwnerAddress' column into 'Address', 'City', and 'State' parts using the PARSENAME function along with REPLACE.

```sql
--4]SPLITTING THE VALUES IN 'OwnerAddress' INTO ADDRESS, CITY, STATE


    ----SHOWCASING THE SPLIT VALUES
    SELECT
    PARSENAME(REPLACE(OwnerAddress,',','.'),3) AS Address,
    PARSENAME(REPLACE(OwnerAddress,',','.'),2) AS City,
    PARSENAME(REPLACE(OwnerAddress,',','.'),1) AS State
    FROM housing;
```

- These queries involve altering the 'housing' table by adding 'Owner_Address', 'Owner_City', and 'Owner_State' columns to store the split values. Then, the existing 'OwnerAddress' column is updated to fill these new columns with the respective address, city, and state parts.

```sql
--4]SPLITTING THE VALUES IN 'OwnerAddress' INTO ADDRESS, CITY, STATE


    ----ADDING COLUMNS 'Address', 'City' & 'State' AND UPDATING THEM WITH THE SPLIT VALUES
    ALTER TABLE housing
        ADD
            Owner_Address nvarchar(255);

    ALTER TABLE housing
        ADD
            Owner_City nvarchar(255);

    ALTER TABLE housing
        ADD
            Owner_State nvarchar(255);


    UPDATE housing
        SET
            Owner_Address =PARSENAME(REPLACE(OwnerAddress,',','.'),3);


    UPDATE housing
        SET
            Owner_City = PARSENAME(REPLACE(OwnerAddress,',','.'),2);


    UPDATE housing
        SET
            Owner_State = PARSENAME(REPLACE(OwnerAddress,',','.'),1);
```

5] UPDATING ABBREVIATIONS :

- This query demonstrates how to transform the 'Y' and 'N' values in the 'SoldAsVacant' column into 'Yes' and 'No', respectively, using a CASE expression.

```sql
--5]UPDATE THE 'Y' & 'N' VALUES IN THE COLUMN "SoldAsVacant" TO 'YES' AND 'NO'

    ----SHOWCASING THE 'Y' & 'N' AS 'Yes' & 'No'
    SELECT SoldAsVacant,
        CASE
            WHEN SoldAsVacant = 'Y' THEN 'Yes'
            WHEN SoldAsVacant = 'N' THEN 'No'
            ELSE SoldAsVacant
        END
    FROM housing;
```

- This query involves updating the 'SoldAsVacant' values in the 'housing' table by using a CASE expression similar to the one shown in the first query. It changes 'Y' to 'Yes' and 'N' to 'No', leaving other values unchanged.

```
--5]UPDATE THE 'Y' & 'N' VALUES IN THE COLUMN "SoldAsVacant" TO 'YES' AND 'NO'

    ----UPDATING THE VALUES TO 'YES' & 'NO'
    UPDATE housing
    SET SoldAsVacant =
        CASE
            WHEN SoldAsVacant = 'Y' THEN 'Yes'
            WHEN SoldAsVacant = 'N' THEN 'No'
            ELSE SoldAsVacant
        END
    FROM housing;
```

6] REMOVING DUPLICATES :

- This query identifies duplicate records in the 'housing' table based on the columns 'ParcelID', 'SaleDate', and 'LegalReference'. It counts the occurrences of each combination of these columns and showcases instances where there are multiple records with the same combination.

```
--6] REMOVING DUPLICATES

    ----SHOWCASING THE DUPLICATE VALUES

    SELECT ParcelID, SaleDate, LegalReference, COUNT(*) AS DuplicateCount
    FROM housing
    GROUP BY ParcelID, SaleDate, LegalReference
    HAVING COUNT(*) > 1;
```

- This query involves removing duplicate records from the 'housing' table. It retains only the records with the minimum 'UniqueID' for each combination of 'ParcelID', 'SaleDate', and 'LegalReference'.

```
--6] REMOVING DUPLICATES

    ----DELETING THE DUPLICATE VALUES

    DELETE FROM housing
    WHERE [UniqueID ] NOT IN (
        SELECT MIN([UniqueID ])
        FROM housing
        GROUP BY ParcelID, SaleDate, LegalReference
    );
```

7] REMOVING UNUSED COLUMNS :

- This query is focused on deleting unused columns from the 'housing' table. The ALTER TABLE statement with the DROP COLUMN clause is used to remove specific columns from a table.

```
 --7]DELETING UNUSED COLUMNS

ALTER TABLE housing
DROP COLUMN PropertyAddress,OwnerAddress,TaxDistrict;
```

```sql
--2]POPULATING PROPERTY ADDRESS DATA

    --SHOWCASES ALL THE NULL ROWS WHICH HAS SAME 'ParcelID' BUT DIFFERENT 'UniqueID' BETWEEN THE TWO COLUMNS

    SELECT a.ParcelID, a.PropertyAddress, b.ParcelID, b.PropertyAddress
    FROM housing AS a
    join
    housing AS b
    ON
        a.ParcelID = b.ParcelID
        and a.[UniqueID ] <> b.[UniqueID ]
    WHERE
        a.PropertyAddress IS NULL;
```