CLEANING EXCEL SPREADHSEETS USING SQL

In my experience learning SQL for data analysis, I have used three different platforms: MySQL, BigQuery, and SQL Server Management Studio. This project uses code specific to the latter, as I wanted to distinguish it from the previous COVID-19 project in which I used BigQuery.

This project examines how to clean data outside of using spreadsheets. While Excel is fun to filter down and remove blanks and duplicates, sometimes datasets are so unwieldy that Excel (or Google Sheets) is insufficient for the task.

This was inspired by Alex the Analyst, in which he provided a more advanced SQL tutorial to do some interesting things. The dataset he/I used can be accessed and downloaded on his Github profile here.

To prepare my data, it was first necessary to install the essential components: SSMS and SQL Server 19. I specifically used the 19th edition, as I found its Import and Export Wizard the most user-friendly. (Technical detail: to connect to the server, I clicked options > connection properties > and checked the "trust server certificate" box.)

Once my tools were setup, I loaded the dataset (in an Excel spreadsheet) as a CSV using the Import Wizard. From there, I was ready to clean this magnificent raw data set of Nashville Housing.

```
select *
from master.dbo.NashvilleHousing;
```

FIRST, STANDARDIZE THE DATE FORMAT SO IT IS EASIER ON THE EYES.

```
select SaleDate, CONVERT(date, SaleDate)
from master.dbo.NashvilleHousing;
Update NashvilleHousing
SET SaleDate = CONVERT(Date,SaleDate)
EXECUTE
ALTER TABLE NashvilleHousing
Add SaleDateConverted Date;

Update NashvilleHousing
SET SaleDateConverted = CONVERT(Date,SaleDate)
```

Execute.

```
select SaleDateConverted, CONVERT(date, SaleDate)
from master.dbo.NashvilleHousing;
```

Run to verify it worked; it did. This cleans the data in the column to be more concise and legible.

POPULATE PROPERTY ADDRESS DATA

```
select PropertyAddress
```

```
from master.dbo.NashvilleHousing
where PropertyAddress is null;
```
Retrieves all null properties in the given column
```
select *
from master.dbo.NashvilleHousing
where PropertyAddress is null;
```

Shows the table with all associated rows where properties are null.

```
select *
from master.dbo.NashvilleHousing
--where PropertyAddress is null;
order by ParcelID;
```

Yields a list of results in which there are duplicate rows of Parcel ID's as well as addresses. The parcel ID is going to be the same as the property address.

```
Select a.ParcelID, a.PropertyAddress, b.ParcelID, b.PropertyAddress,
ISNULL(a.PropertyAddress,b.PropertyAddress)
from master.dbo.NashvilleHousing a
JOIN master.dbo.NashvilleHousing b
	on a.ParcelID = b.ParcelID
	AND a.[UniqueID ] <> b.[UniqueID ]
Where a.PropertyAddress is null;
```

The no column name result is what we want to place in the property address column that is full of null values. In order to do so, we need to run the following code to update the column. We use the alias "a" because with JOINS we cannot use the Nashville housing name.

```
Update a
SET PropertyAddress = ISNULL(a.PropertyAddress,b.PropertyAddress)
from master.dbo.NashvilleHousing a
JOIN master.dbo.NashvilleHousing b
	on a.ParcelID = b.ParcelID
	AND a.[UniqueID ] <> b.[UniqueID ]
Where a.PropertyAddress is null;
```

29 rows affected. We reselect the previous code to verify. Only the row names come up, with no null results. Cleaning successful.

Now, we want to clean the addresses by splitting them up into individual columns for street, city, and state. Let's pull up the data we want to alter with the following code. Here, we can see the entire line is cluttered with all the data, using commas as a delimiter. We want to ultimately separate these out.

```
Select PropertyAddress
from master.dbo.NashvilleHousing
--Where PropertyAddress is null
--order by ParcelID;
```

We remove the delimiter by adding a negative one into the line of code, as illustrated below. The result gives us a row with the house number and the street name.

```
SELECT
```

```
SUBSTRING(PropertyAddress, 1, CHARINDEX(',', PropertyAddress)-1) as Address

from master.dbo.NashvilleHousing;
```

Altering the code with another line, this time with a plus one, separates the city from the street address without the delimiter cluttering the tables up.

```
SELECT
SUBSTRING(PropertyAddress, 1, CHARINDEX(',', PropertyAddress)-1) as Address
, SUBSTRING(PropertyAddress, CHARINDEX(',', PropertyAddress) + 1 , LEN(PropertyAddress))
as Address

from master.dbo.NashvilleHousing;
```

Now, we want to create new columns to house the broken-up data. And we can do that with the following code, each executed one at a time.

```
ALTER TABLE NashvilleHousing
Add PropertySplitAddress Nvarchar(255);

Update NashvilleHousing
SET PropertySplitAddress = SUBSTRING(PropertyAddress, 1, CHARINDEX(',', PropertyAddress)
-1 )


ALTER TABLE NashvilleHousing
Add PropertySplitCity Nvarchar(255);

Update NashvilleHousing
SET PropertySplitCity = SUBSTRING(PropertyAddress, CHARINDEX(',', PropertyAddress) + 1 ,
LEN(PropertyAddress))
```

Finally, to verify it worked, we can view the entire data set. And by scrolling to the very end, we can see that we created new columns. This data is so much more convenient than the original full string.

```
Select *
from master.dbo.NashvilleHousing;
```

Now, let's delete our unused columns from the data we separated earlier. Alex performed this operation near the end of his tutorial, but for my brain it makes more sense to complete the process of cleaning in its entirety prior to moving on to the next data selection. I'm sure either way is acceptable, provided it gets done.

DISCLAIMER: never delete anything in your original dataset unless you are certain that you aren't going to permanently disfigure your file- for you and everyone else.

```
select *
from master.dbo.NashvilleHousing;


ALTER TABLE master.dbo.NashvilleHousing
DROP COLUMN OwnerAddress, TaxDistrict, PropertyAddress, SaleDate;
```

After running the code, select everything again to verify it cleaned the table.

Now, we want to do a similar cleaning task with the owner address column. Running the following code gives us a column that has the address, city, and state.

```
Select OwnerAddress
from master.dbo.NashvilleHousing;
```

We want to split all of these out. Instead of using substrings, we can use parse name. This breaks the data down into separate columns for street, city, and state respectively.

```
Select
PARSENAME(REPLACE(OwnerAddress, ',', '.') , 3)
,PARSENAME(REPLACE(OwnerAddress, ',', '.') , 2)
,PARSENAME(REPLACE(OwnerAddress, ',', '.') , 1)
from master.dbo.NashvilleHousing;
```

Now that we have our separate data, as before, we need to add our columns and values. Run each of the following one at a time, then select all for visual purposes and verify that it executed properly.

```
ALTER TABLE NashvilleHousing
Add OwnerSplitAddress Nvarchar(255);

Update NashvilleHousing
SET OwnerSplitAddress = PARSENAME(REPLACE(OwnerAddress, ',', '.') , 3)


ALTER TABLE NashvilleHousing
Add OwnerSplitCity Nvarchar(255);

Update NashvilleHousing
SET OwnerSplitCity = PARSENAME(REPLACE(OwnerAddress, ',', '.') , 2)

ALTER TABLE NashvilleHousing
Add OwnerSplitState Nvarchar(255);

Update NashvilleHousing
SET OwnerSplitState = PARSENAME(REPLACE(OwnerAddress, ',', '.') , 1)

Select *
from master.dbo.NashvilleHousing;
```

And there we have it. Three new columns with the geographical data separated by street name, city, and state.

Let’s now look at the sold as vacant column.

```
Select Distinct(SoldAsVacant), Count(SoldAsVacant)
from master.dbo.NashvilleHousing
Group by SoldAsVacant
order by 2;
```

This gives us a couple of things. Two yes/no columns, specified by both “Y”, “N”, and “Yes”, “No”. Yes and No have far more values than Y and N, and we want to consolidate for simplicity.

```
Select SoldAsVacant
, CASE When SoldAsVacant = 'Y' THEN 'Yes'
          When SoldAsVacant = 'N' THEN 'No'
          ELSE SoldAsVacant
          END
from master.dbo.NashvilleHousing;
```

The result demonstrates where single-character responses were transformed into full word responses.

```
Update NashvilleHousing
SET SoldAsVacant = CASE When SoldAsVacant = 'Y' THEN 'Yes'
          When SoldAsVacant = 'N' THEN 'No'
          ELSE SoldAsVacant
          END;
```

This updated 56,477 rows. Let's rerun the original selection to verify.

```
Select Distinct(SoldAsVacant), Count(SoldAsVacant)
from master.dbo.NashvilleHousing
Group by SoldAsVacant
order by 2;
```

It worked! Now we have two columns, one for "Yes" and one for "No", having consolidated the "Y" and "N" responses.

Now let's turn our attention to removing duplicates from the table. While it is not standard practice to delete data from the database, we are going to try this for illustrative purposes. (We would ordinarily never want to delete anything from workplace, real-world datasets, as it's very easy to make permanent mistakes. To reiterate from earlier: don't try this at home.) We will do a CTE to find where there are duplicate values, and then delete them.

```
WITH RowNumCTE AS(
Select *,
       ROW_NUMBER() OVER (
       PARTITION BY ParcelID,
                            PropertyAddress,
                            SalePrice,
                            SaleDate,
                            LegalReference
                            ORDER BY
                                  UniqueID
                                  ) row_num

from master.dbo.NashvilleHousing
--order by ParcelID
)
DELETE
From RowNumCTE
Where row_num > 1
--Order by PropertyAddress;
```

And there we have it. This CTE deleted 104 rows of duplicate data successfully. We rerun the code with a select all command, and POOF! There are no more rows of duplicates.

```
WITH RowNumCTE AS(
```

```
Select *,
       ROW_NUMBER() OVER (
       PARTITION BY ParcelID,
                            PropertyAddress,
                            SalePrice,
                            SaleDate,
                            LegalReference
                            ORDER BY
                                  UniqueID
                                  ) row_num

from master.dbo.NashvilleHousing
--order by ParcelID
)
select *
From RowNumCTE
Where row_num > 1
--Order by PropertyAddress;
```

It worked! The data is now clean, and significantly more user-friendly for further analysis as needed.