Data cleaning is an essential step in data analysis as it helps ensure that the data used is accurate, reliable, and consistent. In this article, we will explore different cases of data cleaning using SQL, specifically using PostgreSQL.

1. Handling missing or NULL values, which can be dealt with by removing the rows with missing values, filling in some fixed errors, or inputting a calculated value. We will also discuss factors to consider when deciding whether to remove rows or keep them. Ultimately, by following best practices for data cleaning, we can improve the quality and accuracy of our analysis.

```
DELETE FROM table_name WHERE column_name IS NULL;
```

2. **Filling the data set with a default value**
   When there are just a few missing or NULL data and their values are not important to the analysis, one alternative is to fill in with a default value. In case, you have prior knowledge or understanding of the missing fields in the data, or some systematic errors, you can also decide to fill in with a default value.

```
SELECT COALESCE(column_name, 'unknown') FROM table_name;
```

The function COALESCE() will return the value in column_name, but if it is NULL, it will return the string “unknown” instead.

3. **Filling the data set with the values from the previous or next rows**Consider the scenario where some of the stock prices in the “price” column of the “stock” table are missing. We can use the LAG()and LEAD()functions to fill in these missing values based on the price values of the corresponding stock on the next or previous day. For example:

```
-- get the price of the next day
UPDATE stock_data
SET price = LEAD(price) OVER (PARTITION BY stock ORDER BY date)
WHERE price IS NULL;
```

4. **Removing Duplicates :** I usually use ROW_NUMBER() to rank the rows. If there are rows with the same rank, it means the rows are duplicated. To remove the duplications, I often keep the first row only.

```
-- Identify and handle duplicate rows by keeping only the first occurrence
SELECT column1, column2, ...
FROM (
  SELECT column1, column2, ..., ROW_NUMBER() OVER (PARTITION BY column1, column2
  FROM table_name
) t
WHERE rn = 1;
```

One thing to notice when removing duplicates is the data types of the columns being compared. For example, when comparing string values, attention should be paid to **case sensitivity and leading or trailing spaces**. “us bank” and “US Bank” are considered two different cases in SQL, and the same goes for “USBank” and “US Bank”. This leads us to the third step of cleaning data, which is **standardizing values.**

5. **Convert string to a date data type :** Sometimes, the date is saved in string values. To bring its format back to data type, consider using TO_DATE() function as below:

```
SELECT TO_DATE(date_column, 'MM/DD/YYYY') AS new_date
FROM table_name;
```

6. et's say you have a date column in the format MM-DD-YYYY, and you need to convert that column into a different string format DD/MM/YYYY.

```
SELECT TO_CHAR(TO_DATE(date_column, 'MM-DD-YYYY'), 'DD/MM/YYYY') AS new_formatte
FROM table_name;
```

7. **Remove leading/trailing spaces**

In this case, we can use the function TRIM() to remove any leading or trailing spaces in the data column.

```
SELECT DISTINCT ON (LOWER(TRIM(column_name))) *
FROM table_name
ORDER BY LOWER(TRIM(column_name);
```

8. **Convert units of measurements :** Let’s say you wish to convert a column whose lengths are given in feet to meters:

```
SELECT length_column / 3.281 AS standardized_length
FROM table_name;
```

9. **Standardize currency values**

Suppose multiple different currency formats are recorded in the column money. For example, $23, 33 USD, or 43.00. Here is one way of standardizing them with CASE WHEN ()

```
SELECT
  CASE
    WHEN currency_column LIKE '$%' THEN REPLACE(currency_column, '$', '')::numer
    WHEN currency_column LIKE '%USD%' THEN REPLACE(currency_column, ' USD', ''):
    ELSE currency_column::numeric
  END AS standardized_currency
FROM table_name;
```

10. **Standardize misspelled words :** There are many cases when the column contains records with abnormal characters. To transform inconsistent values, we can try using a regular expression to negate the ones that are not letters or digits, then replace them.

```
SELECT REGEXP_REPLACE(column, '[^a-zA-Z0-9]', '') AS transformed_column1
FROM table_name;
```

11. Or, to put it more simply, if you frequently encounter special characters in text values, you can use the REPLACE()function to remove or replace them.

```
-- Remove special characters from text values
SELECT REPLACE(column1, ',', '') AS cleaned_column1
FROM table_name;

-- Correct spelling errors in text values
SELECT REPLACE(column1, 'color', 'Colour') AS corrected_column1
FROM table_name;
```

12. **Handling Outliers**

Calculating the % change between consecutive rows in a column

To discover any outliers or inaccuracies in the data, it might be useful to calculate the percentage change between consecutive rows in a

column. For instance, a data entry error might be indicated by a percentage change that is significantly greater or lower than expected.

```
SELECT (column1 - LAG(column1) OVER (ORDER BY id)) / LAG(column1) OVER (ORDER BY
FROM table_name;
```

13. **Removing or Transforming :** After detecting the outliers in the data profiling step, we need to consider removing, transforming, or replacing them with suitable values. Finding reasonable values to replace depends on your data knowledge as well as your domain understanding. Here is the code that you may find useful to apply in some situations of outliers.

```
-- Remove rows with values outside a specified range
SELECT column1, column2, ...
FROM table_name
WHERE column1 BETWEEN lower_limit AND upper_limit;

-- Transform outliers using a logarithmic function
SELECT LOG(column1) AS transformed_column1
FROM table_name;
```

14. **Handling Text Values**

In case you want to concatenate the columns of “last name” and “first name” in a column called “full name”, you may consider using CONCAT() :

```
SELECT CONCAT(firstname, ' ', lastname) AS full_name
FROM table_name;
```

To maintain the accuracy and reliability of data, data cleaning should be a continuous process rather than a one-time task. Regularly performing data cleaning tasks can help prevent errors and inconsistencies from accumulating over time, leading to more accurate and trustworthy data.