SQL: NULL CHECKING

`NULL` is a special value that represent missing data. Any operation on a `NULL` value will result in a `NULL` value not matter which one.

`NULL` is a touchy subject. Indeed, there are no real rules that enforce to use or not the `NULL` value. However we can restore to basic guidelines in order to avoid problems later on:

Summary

- **Use `NULL` when**:
    - The data is optional.
    - The data might be unknown or not available yet.
    - The absence of data is meaningful and significant.
    - **Good Practices for Using `NULL` in SQL**

        1. **Understand the Purpose of `NULL`**:
            - `NULL` represents the absence of a value or an unknown value. It is not the same as zero or an empty string.
        2. **Default Values**:
            - Whenever possible, avoid `NULL` by providing default values. This helps in avoiding `NULL` checks in queries.
            - Example: Use DEFAULT 0 for numerical fields, DEFAULT '' for strings, or appropriate default values for other types.
        3. **Use NOT NULL Constraints**:
            - Apply `NOT NULL` constraints on columns that must have a value. This enforces data integrity.
            - Example:

            ```
            CREATE TABLE users (id INT PRIMARY KEY, name VARCHAR
            ```

            This table creates a `user` table where we state the `id` column is a primary key with a limit on the number of character (up to 100) and it must be filled-in !

4. **Careful Use in Conditions**:
    - Be cautious when using `NULL` in conditions as it can lead to unexpected results.
    - Use `IS NULL` or `IS NOT NULL` instead of = or != for comparisons.
    - Example: Among all the columns, return all the lines where there is no name (NULL)

      ```
      SELECT * FROM users WHERE name IS NULL;
      ```

5. **Functions and Aggregates**:
    - Be aware of how SQL functions and aggregate functions handle `NULL`.
    - For example, `COUNT(column)` ignores `NULL` values, whereas `SUM(column)` returns `NULL` if all values are `NULL`.
    - Use functions like `COALESCE` to handle `NULL` values. The `COALESCE` function returns the first non `NULL` value in a list. We dedicate a section to this one.
6. **COALESCE and ISNULL**:
    - Use `COALESCE` or `ISNULL` (or equivalent) to provide fallback values for `NULL`.
    - Example:

      ```
      SELECT COALESCE(name, 'Unknown') FROM users;
      ```

7. **Avoid NULL in Primary Keys**:
    - Primary keys should never be `NULL` as they uniquely identify records in a table.
    - Always define primary key columns with `NOT NULL`.
8. **NULL vs. Empty String**:
    - Decide whether to use `NULL` or an empty string for missing text data and be consistent throughout the database design.
    - Avoid mixing `NULL` and empty strings to represent missing data in the same column.

9. **Indexing Considerations**:
    - Be aware that `NULL` values are indexed differently depending on the database system. Check your DBMS documentation on how it handles NULL values in indexes.
10. **Consistent Usage**:
    - Establish and follow consistent guidelines within your team or organization for when and how to use `NULL`.
    - Document these guidelines to ensure everyone understands and adheres to them.

- **Avoid NULL when**:
    - The field is required and should always have a value.
    - The field represents a boolean value or binary state.
    - Sensible default values can be provided.
    - The field is a primary key or generally should be a non-null foreign key.

We can check for `NULL` values like this:

```
SELECT column1, column2, ... FROM table_name WHERE column_3
IS NULL;
```

This `NULL` adds upon the two Boolean state true of false and therefore we speak of **three-valued logic.**

Example Table: Customers

customer_id	first_name	last_name	email
1	John	Doe	john.doe@example.com
2	Jane	Smith	NULL
3	Alice	Johnson	alice.j@example.com
4	Bob	Brown	NULL

1. **Checking for `NULL` Values**:

    To select all customers who do not have an email address (`NULL` values in the `email` column):

```
SELECT customer_id, first_name, last_name
FROM Customers
WHERE email IS NULL;
```

Result:

customer_id	first_name	last_name
2	Jane	Smith
4	Bob	Brown

This query returns only those customers who have `NULL` values in the `email` column.

2. **Checking for Non-`NULL` Values**:

   To select all customers who have an email address (non-`NULL` values in the `email` column):

```
SELECT customer_id, first_name, last_name, email
FROM Customers
WHERE email IS NOT NULL;
```

Result:

customer_id	first_name	last_name	email
1	John	Doe	john.doe@example.com
3	Alice	Johnson	alice.j@example.com

This query returns only those customers who have non-`NULL` values in the `email` column.

Key Points

- **`NULL` is Not Equal to Any Value**: In SQL, `NULL` is not equal to any other value, not even another `NULL`. This is why you can't use `=` to check for `NULL`. Instead, you must use `IS NULL` or `IS NOT NULL`.
- **Special Handling of `NULL` in Conditions**: Always use `IS NULL` or `IS NOT NULL` when filtering for `NULL` values to ensure your conditions are evaluated correctly.

- `NULL` **in Other SQL Clauses**: `NULL` values are also handled in other SQL operations and functions, such as aggregate functions (`COUNT`, `SUM`, etc.), `GROUP BY`, and `JOIN` operations, often needing special consideration.

Using `IS NULL` and `IS NOT NULL` allows you to handle missing or unknown data effectively, ensuring that your queries return the precise results you need.