Limpieza de Datos en SQL: De la inconsistencia a la precisión

Como entusiasta de SQL, solía navegar por plataformas de bases de datos como Kaggle en busca de conjuntos de datos limpios y listos para el análisis. Sin embargo, me di cuenta de que el 90% de las veces que encontraba un conjunto interesante, me enfrentaba a datos crudos que presentaban numerosos problemas: nombres de columnas inapropiados, fechas y cifras en formato de texto, caracteres extraños que complicaban las consultas simples, registros vacíos, y espacios innecesarios, como los que aparecen antes de un nombre o en dobles espacios dentro de áreas administrativas, como “Recursos humanos”.

Este panorama me llevó a replantear mis objetivos con SQL. En lugar de enfocarme únicamente en realizar análisis complejos, consultas avanzadas y joins, decidí priorizar un aspecto fundamental: la limpieza de datos. Reconocí que, antes de hacer cualquier análisis significativo, debía asegurarme de que los datos estuvieran en buen estado.

Uno de los problemas más comunes que encontré fue con las columnas de fechas, que a menudo estaban en formato de texto. Esto imposibilitaba realizar análisis básicos, como calcular el total de ventas por mes o determinar la antigüedad de un empleado. En mi primer artículo de esta serie, abordé este tema porque fue una de mis principales prioridades y, honestamente, lo que más me costó dominar. Al enfrentarlo primero, pude construir una base sólida para mis proyectos futuros.

Otro desafío frecuente se presentaba en columnas como salario, ventas totales e ingresos, que mostraban inconsistencias debido a símbolos de monedas, comas, puntos y, por supuesto, estaban en formato de texto. Estas irregularidades dificultaban análisis elementales, como calcular salarios por área o identificar los meses con mayores ingresos.

Adicionalmente, también me encontré lidiando con múltiples espacios en los registros, como “ Ricardo”, “Recursos humanos”, “Adriana “. Aprender a resolver estos problemas no solo fue gratificante, sino que descubrí que tengo una inclinación natural hacia la organización y el análisis (al menos en mi trabajo jajaja).

Con este artículo, no solo quiero mostrar el proceso técnico que conlleva la limpieza de datos, sino también ayudar a aquellos que se sienten atraídos por SQL y estan considerando iniciar proyectos enfocados en la limpieza y normalización de bases de datos.

Pasos Esenciales para una Limpieza de Datos Efectiva

A continuación, comparto algunos pasos que considero esenciales para llevar a cabo una limpieza de datos efectiva:

1. **Convertir el archivo en tabla**: Generalmente, uso Excel para importar los datos correctamente a SQL, asegurándome de que el tipo de datos sea el adecuado durante la importación.
2. **Revisar la ortografía**: La consistencia es clave. Es importante verificar que no haya variaciones en los nombres de los campos.
3. **Limpiar espacios**: Además de eliminar espacios al inicio y al final, se deben buscar y eliminar espacios adicionales entre palabras.
4. **Ajustar nombres de columnas**: Es fundamental unificar el idioma y estilo de los nombres de las columnas. Mantener una convención de nombres clara ayudará en futuras consultas.
5. **Formato a cifras o números**: Asegúrate de que las columnas con datos numéricos tengan el tipo de dato correcto y que no contengan caracteres no numéricos.
6. **Formato de fechas**: Unificar el formato de las fechas es crucial para cualquier análisis temporal.
7. **Renombrar booleanos**: Asegúrate de utilizar etiquetas claras y consistentes para valores booleanos.
8. **Manejo de celdas vacías**: Detecta y decide cómo manejar las celdas vacías de manera efectiva.
9. **Interpretación de valores nulos**: Es fundamental saber cómo manejar y tratar los valores nulos basándose en el contexto de los datos.
10. **Control de duplicados**: Verificar y eliminar registros duplicados es esencial para mantener la integridad de los datos.
11. **Documentación**: Llevar un registro de los cambios y los pasos que se siguen durante el proceso de limpieza no solo facilitará el trabajo futuro, sino que también asegurará la reproducibilidad.

1. Creación de la Base de Datos y Procedimiento Almacenado

Comencé creando una base de datos llamada *'clean'* donde almacené la tabla limpieza. Para facilitar la reutilización de ciertas consultas, creé un procedimiento almacenado simple llamado *limp()* que selecciona todos los registros de la tabla. Esto me permite verificar fácilmente los cambios en los datos a lo largo del proceso de limpieza:

```
CREATE DATABASE clean;

use clean;

SELECT * FROM limpieza;


DELIMITER //
CREATE PROCEDURE limp()
BEGIN
        select * from limpieza;
END //

DELIMITER ;

CALL limp();
```

Este enfoque ayuda a optimizar el flujo de trabajo, permitiendo verificar resultados en cada paso del proceso.

2. Renombrar Columnas de la Tabla

Uno de los primeros pasos en la limpieza de datos es asegurarse de que los nombres de las columnas sean claros y consistentes. A menudo, los nombres vienen con caracteres especiales o formateados incorrectamente. En mi caso, renombré varias columnas utilizando la siguiente sintaxis:

```
-- renombro las columnas

SET SQL_SAFE_UPDATES = 0;

ALTER TABLE limpieza CHANGE COLUMN `ï»¿Id?empleado` `id_emp` VARCHAR(20) NULL;
ALTER TABLE limpieza CHANGE COLUMN `gÃ©nero` `gender` varchar(20) null;
ALTER TABLE limpieza CHANGE COLUMN Apellido last_name varchar(50) null;
ALTER TABLE limpieza CHANGE COLUMN star_date start_date varchar(50) null;
ALTER TABLE limpieza CHANGE COLUMN Name name varchar(50) null;
```

Al realizar este cambio, mejoré la claridad de los nombres y se garantiza que los futuros análisis sobre los datos fueran más legibles y entendibles. También me aseguré de que los tipos de datos fueran correctos y que las columnas aceptaran valores NULL cuando fuera necesario.

El comando *'SET SQL_SAFE_UPDATES = 0'* desactiva el modo de actualizaciones seguras en MySQL. Este modo, cuando está activado (su valor es 1), impide realizar UPDATE o DELETE sin una clausula WHERE o sin una clave primaria, evitando que se modifiquen o borren registros accidentalmente. Por lo tanto, al desactivarlo nos permite realizar actualizaciones a toda la tabla, como renombrar columnas o actualizar registros sin necesidad de especificar una condición exacta.

3. Identificación y Eliminación de Duplicados

La limpieza de datos no estaría completa si no abordamos los registros duplicados. En este caso, identifiqué los duplicados utilizando una subconsulta con COUNT y GROUP BY. Luego, utilicé una CTE (Expresión Común de Tabla) para hacerlo más eficiente:

```
WITH total_duplicados AS (
    SELECT id_emp, COUNT(*) AS duplicados
    FROM limpieza
    GROUP BY id_emp
    HAVING COUNT(*) > 1
)
SELECT COUNT(*) AS cantidad_duplicados
FROM total_duplicados;

# 9 duplicados
```

Encontré que había 9 registros duplicados. Para eliminarlos, creé una tabla temporal sin duplicados y reemplacé la tabla original:

```
rename table limpieza to conduplicados;

-- creo una tabla temporal sin duplicados
create temporary table  templimpieza AS
SELECT DISTINCT * FROM conduplicados;

select count(*) as original FROM conduplicados;
-- 22223

SELECT COUNT(*) AS copia FROM templimpieza;
-- '22214'

-- como se ve, la diferencia es de 9 registros con duplicados

CREATE TABLE limpieza as
select * from templimpieza;

-- finalmente elimino la tabla con duplicados

DROP TABLE conduplicados;
```

Este paso fue crucial para asegurar la integridad de los datos, ya que los registros duplicados pueden sesgar los análisis y provocar resultados erróneos. Aquí se usan CTEs, para profundizar sobre ellas los invito a leer mi artículo que muestra la diferencia con las Subqueries. (https://www.linkedin.com/pulse/ctes-vs-subqueries-comparativa-y-practicas-para-en-sql-olascoaga-zgiuc/?trackingId=TGX%2BeTgNQJWOn7ZpV3UNZA%3D%3D)

4. Eliminación de Espacios en Blanco

Una de las cuestiones más comunes en la limpieza de datos es el manejo de espacios en blanco innecesarios. Para esto, primero utilicé la función *TRIM()* para seleccionar aquellos nombres que contenían espacios antes y despues del nombre. Posteriormente, luego de consultar y verificar que existían espacios, procedí a eliminar los espacios al principio y al final de las cadenas de texto:

```
SELECT TRIM(name) as name from limpieza;
SELECT TRIM(last_name) as last_name from limpieza;

SELECT name, trim(name) FROM limpieza
WHERE length(name) - length(trim(name)) > 0;

with espacios as (
SELECT TRIM(name) as name from limpieza)
SELECT name FROM espacios
WHERE length(name) - length(trim(name)) > 0

-- cambio permanentenmente la columna name

UPDATE limpieza SET name = trim(name);
```

Hago lo mismo con la columna 'last_name':

```
-- hago lo mismo con la columna apellidos

SELECT last_name, trim(last_name)
FROM limpieza
WHERE length(last_name) - length(trim(last_name)) > 0;

-- verifico con una CTE que la query funcione para cambiar la tabla

WITH espacios_last_name AS (
SELECT TRIM(last_name) as last_name FROM limpieza)
SELECT length(last_name) - length(trim(last_name)) > 0 AS espacios
FROM espacios_last_name;

UPDATE limpieza SET last_name = trim(last_name);
```

Ahora, un caso habitual que tambien podría ocurrir, es que haya más de un espacio entre los registros en una columna, como podría ser el ejemplo de "Recursos Humanos". Estos errores pueden generar inconsistencias en el análisis y visualización de datos, por lo que es crucial eliminarlos de manera eficiente.

Para identificar y eliminar estos espacios en blanco adicionales, utilicé una combinación de expresiones regulares y la función *REGEXP_REPLACE.*

Identificación de Espacios Dobles con Expresiones Regulares

El primer paso fue identificar las filas que contenían dos o más espacios consecutivos en la columna área. Para esto, utilicé una expresión regular con la función REGEXP, que me permite buscar patrones específicos dentro del texto. Aquí, *\\s* representa cualquier espacio en blanco (incluyendo espacios y tabulaciones), y *{2,}* especifica que quiero buscar dos o más espacios consecutivos:

```
SELECT area from limpieza
WHERE area regexp '\\s{2,}';
```

```
SELECT area, trim(regexp_replace(area,'\\s{2,}',' ')) as ensayo
FROM limpieza;

UPDATE limpieza SET area = trim(regexp_replace(area,'\\s{2,}',' '));
```

REGEXP: Esta función permite realizar búsquedas avanzadas basadas en patrones definidos por expresiones regulares.

\\s{2,}: En esta expresión, \\s es un carácter especial que representa cualquier tipo de espacio en blanco, mientras que {2,} indica que buscamos dos o más espacios consecutivos.

Una vez identificados los registros con espacios dobles, el siguiente paso sería corregir el problema. Para esto, hay que utilizar la función *REGEXP_REPLACE,* que permite reemplazar patrones específicos dentro de las cadenas de texto. En este caso, reemplacé los múltiples espacios consecutivos por un solo espacio.

5. Normalización de Valores

Normalicé los valores de la columna *gender,* que contenían datos en español, a valores en inglés. Utilicé una combinación de la cláusula *CASE* para realizar este cambio:

```
SELECT gender,
CASE
        WHEN gender = 'hombre' THEN 'male'
    WHEN gender = 'mujer' THEN 'female'
    ELSE 'Other'
END AS gender1
FROM limpieza;

UPDATE limpieza SET gender = CASE WHEN gender = 'hombre' THEN 'male'
    WHEN gender = 'mujer' THEN 'female'
    ELSE 'Other'
END;

CALL limp();
```

6. Convertir Indicadores Binarios en Descripciones Claras

Uno de los aspectos más importantes al preparar los datos para el análisis es asegurarse de que sean comprensibles para quienes los usarán. En la columna type, que indicaba si un empleado trabajaba de forma remota o híbrida, los valores eran *0 y 1,* lo que puede resultar confuso. Para solucionarlo, convertí estos indicadores numéricos en descripciones más claras:

```
-- la columna type me indica si el empleado trabaja remoto o hibrido
-- por lo tanto paso esa columna a formato texto para realizar el siguiente cambio
-- ya que 0 y 1 es confuso para quien llevara a cabo el analisis

ALTER TABLE limpieza modify column type TEXT;

SELECT type,
CASE
        WHEN type = 0 THEN "remote"
    WHEN type = 1 THEN "hybrid"
    ELSE "other"
END AS ejemplo
FROM limpieza;


UPDATE limpieza
SET type =
CASE
        WHEN type = 1 THEN 'remote'
    WHEN type = 0 THEN 'hybrid'
    ELSE 'other'
END;

call limp();
```

Modificación de tipo de dato: Cambié el tipo de la columna type a TEXT para facilitar la inserción de valores textuales.

Uso de CASE: Utilicé la cláusula CASE para transformar los valores 0 y 1 en descripciones legibles (remote y hybrid), haciendo que los datos sean más fáciles de interpretar para los analistas.

7. Limpieza y Conversión de Datos de Salario

Los valores de salario en la columna *salary* estaban formateados con signos de dólar y comas, lo que no es adecuado para realizar cálculos o análisis financieros. Para limpiar y convertir estos valores a un formato numérico adecuado, utilicé una combinación de funciones:

```
SELECT salary,
                        REPLACE(salary, '$',''),
            REPLACE(salary, ',',''),
            TRIM(salary) as salario
FROM limpieza;
```

```
SELECT salary,
                    CAST(TRIM(REPLACE(REPLACE(salary,'$',''),',','')) AS DECIMAL(15,2)) AS salary_new
FROM limpieza;

-- AHORA SI, CAMBIO LA COLUMNA PERMANENTEMENTE

UPDATE limpieza SET salary =  CAST(TRIM(REPLACE(REPLACE(salary,'$',''),',','')) AS DECIMAL(15,2));

call limp();

alter table limpieza modify column salary int null;
```

Explicación:

REPLACE: Elimino el signo de dólar ($) y las comas (,) que podrían interferir con el formato numérico.

TRIM: Aseguro que no queden espacios en blanco al principio o final de los valores.

CAST: Convierto el texto resultante en un valor decimal con hasta dos posiciones decimales, usando DECIMAL(15,2).

Finalmente, ajusté el tipo de dato de la columna a INT para asegurar que los valores pudieran ser usados en cálculos financieros sin problemas:

8. Estandarización de Fechas

El manejo de fechas es fundamental para muchos análisis (https://www.linkedin.com/pulse/desaf%C3%ADos-en-el-manejo-de-fechas-sql-un-caso-pr%C3%A1ctico-con-olascoaga-ttx7f/?trackingId=Wvliom88Q4qkwvatjTSeQQ%3D%3D), por lo que estandaricé las columnas de fechas que estaban en distintos formatos. Primero trabajé con la columna *birth_date*, donde los formatos incluían tantos guiones (-) como barras diagonales (/):

```
CALL limp();
 SELECT birth_date FROM limpieza;

 -- voy a darle formato año, mes, dia.
 -- uso dos funciones stringtodate, dateformat

 SELECT birth_date, CASE
                WHEN birth_date like '%/%' THEN date_format(str_to_date(birth_date, '%m/%d/%y'),'%y-%m-%d')
          WHEN birth_date like '%-%' THEN date_format(str_to_date(birth_date, '%m-%d-%y'),'%y-%m-%d')
          ELSE null
        END AS new_birth_date
FROM limpieza;
```

```
UPDATE limpieza
SET birth_date = CASE
    WHEN birth_date LIKE '%/%' THEN
        DATE_FORMAT(STR_TO_DATE(birth_date, '%m/%d/%Y'), '%Y-%m-%d')
    WHEN birth_date LIKE '%-%' THEN
        DATE_FORMAT(STR_TO_DATE(birth_date, '%m-%d-%Y'), '%Y-%m-%d')
    ELSE
        NULL
END;

ALTER TABLE limpieza modify column birth_date date;
```

STR_TO_DATE: Convertí las cadenas de texto en valores de fecha mediante el formato adecuado, dependiendo de si contenían barras o guiones.

DATE_FORMAT: Apliqué un formato estándar YYYY-MM-DD para unificar todas las fechas.

Después de asegurarme de que los valores estaban correctamente formateados, modifiqué el tipo de dato de la columna *birth_date* para que sea de tipo DATE.

El mismo proceso fue aplicado a la columna *start_date,* estandarizando también los formatos y modificando el tipo de dato:

```
UPDATE limpieza
SET start_date = CASE
    WHEN start_date LIKE '%/%' THEN
        DATE_FORMAT(STR_TO_DATE(start_date, '%m/%d/%Y'), '%Y-%m-%d')
    WHEN start_date LIKE '%-%' THEN
        DATE_FORMAT(STR_TO_DATE(start_date, '%m-%d-%Y'), '%Y-%m-%d')
    ELSE
        NULL
END;

ALTER TABLE limpieza modify column start_date date;
```

9. Manejo de Fechas y Tiempos en finish_date

La columna *finish_date* contenía tanto fechas como horas, y para facilitar su análisis, decidí separar estos componentes en dos columnas: una para la fecha y otra para la hora.

Primero, realicé una copia de seguridad de la columna original:

```
-- copia de seguridad de finish_date

ALTER TABLE limpieza ADD COLUMN date_backup text;

UPDATE limpieza set date_backup = finish_date;

-- ya tenemos nuestra copia de respaldo
```

Luego, convertí el formato de *finish_date* y separé la fecha de la hora:

```
-- ya tenemos nuestra copia de respaldo

SELECT finish_date, str_to_date(finish_date, '%Y-%m-%d %H:%i:%s') AS fecha from limpieza;

UPDATE limpieza SET finish_date = str_to_date(finish_date, '%Y-%m-%d %H:%i:%s')
WHERE finish_date <> '';

UPDATE limpieza
SET finish_date = STR_TO_DATE(REPLACE(finish_date, ' UTC', ''), '%Y-%m-%d %H:%i:%s')
WHERE finish_date <> '';

call limp();
```

Separación de fecha y hora: Utilicé las funciones *DATE() y TIME()* para extraer la parte correspondiente de la columna *finish_date,* permitiendo realizar análisis más específicos sobre estos valores.

Finalmente, convertí los espacios en blanco en valores nulos para evitar errores futuros en el análisis:

```
-- como en nuestra columna tenemos espacios en blanco cuando hagamos una actualizacion nos arrojara error
-- convierto los espacios en blancos en valores nulos

 UPDATE limpieza SET finish_date = null where finish_date = '';

ALTER TABLE limpieza modify column finish_date datetime;

DESCRIBE limpieza;
```

Conclusión:

Particularmente, este proyecto de limpieza me ha sido una excelente oportunidad para profundizar en el entendimiento de los desafíos que implica trabajar con los datos en bruto. A partir de un conjunto de datos inconsistentes y errores comunes, como nombres de columnas inapropiados, fechas de formatos incorrectos, valores booleanos no tan fácil de interpretar para realizar visualizaciones o análisis. Este proceso no solo me permitió desarrollar habilidades avanzadas y demostrar los conocimientos que desarrolle en artículos anteriores con CTEs y la manipulación de columnas con registros de fechas, sino

que tambien me ayudo a mejorar mi capacidad para abordar problemas complejos siendo mucho más eficiente a la hora de solucionarlos.

Al trabajar en aspectos clave como la normalización de fechas, la conversión de cifras con símbolos de moneda a valores numéricos, el manejo de espacios innecesarios mediante expresiones regulares, aprendí a apreciar la importancia de una base de datos limpia y estructurada.

Por último, este enfoque refuerza la noción de que aunque los análisis complejos y las consultas avanzadas son importantes, ninguna de esas actividades tiene sentido si no trabajamos primero en la integridad y calidad de los datos que vamos a utilizar. La limpieza de datos es un paso fundamental en el proceso de análisis, y en mi caso ha sido una habilidad que me ha ayudado a desarrollar una mayor comprensión y control sobre el flujo de trabajo en SQL.