Francisco Olascoaga

21/09/2024

Desafíos en el Manejo de Fechas en SQL:

Un Caso Práctico con Datos de Empleados

Introducción

Como parte de mi desarrollo en SQL, he estado trabajando con una base de datos que contiene información valiosa sobre empleados. Este conjunto de datos incluye columnas criticas como salario, fechas de cumpleaños, fecha de inicio y fecha de finalización. Durante este proceso, enfrente varios desafíos en la gestión de las fechas y los datos en general, y me gustaría compartir como los aborde.

1. Estandarización de nombres de columnas

Una de las primeras tareas que lleve a cabo fue la limpieza de las columnas para asegurar la coherencia y facilitar futuras consultas. Esto es crucial en la gestión de bases de datos, ya que nombres confusos pueden llevar a errores en las consultas. Realice los siguientes cambios en la tabla:

```
72 •   ALTER TABLE limpieza CHANGE COLUMN `Id?empleado` id_emp varchar(20);
73 •   ALTER TABLE limpieza CHANGE COLUMN `Name` name varchar(20);
74 •   ALTER TABLE limpieza CHANGE COLUMN `Apellido` last_name VARCHAR(20);
75 •   ALTER TABLE limpieza CHANGE COLUMN `género` gender VARCHAR(20);
```

Al renombrar las columnas, mejoré la legibilidad de la base de datos, facilitando la interacción con los datos y reduciendo el riesgo de cometer errores en futuras consultas.

2. Manejo de Fechas: Birth_date y Start_date

Una vez que los nombres de las columnas estaban estandarizados, me enfoque en trabajar con las columnas de fecha de nacimiento (birth_date) y fecha de inicio (start_date). Ambas columnas contenían valores en diferentes formatos, y en algunos casos, contenían valores nulos.

El primer paso fue asegurarme de que los datos estuvieran en un formato de fecha adecuado. Utilice STR_TO_DATE para transformar las fechas que estaban en formato texto a un formato estándar:

```
12  SELECT
13      CASE
14          WHEN SUBSTRING_INDEX(birth_date, '/', 1) > 12 THEN STR_TO_DATE(birth_date, '%d/%m/%Y')
15          ELSE STR_TO_DATE(birth_date, '%m/%d/%Y')
16      END AS Birth_date_formatted
17  FROM limpieza;
```

En esta primera parte, utilizamos una consulta SELECT para transformar los valores de birth_date en un formato de fecha adecuado.

- **SUBSTRING_INDEX(birth_date, '/', 1)**: Este fragmento extrae el primer componente de la fecha (el día o el mes) al dividir la cadena birth_date en base al carácter /.
- **CASE**: Aquí se utiliza una declaración CASE para determinar el formato de fecha. Si el primer componente (día o mes) es mayor que 12, asumimos que el formato es DD/MM/YYYY y lo convertimos usando STR_TO_DATE con el formato correspondiente. Si no, se asume el formato MM/DD/YYYY.
- **STR_TO_DATE**: Esta función convierte la cadena en un objeto de fecha, utilizando el formato especificado.

El resultado de esta consulta muestra cómo se verían los datos formateados, pero no se guarda en la base de datos en este paso.

```
19  ALTER TABLE limpieza ADD COLUMN Birth_date_formatted DATE;
20
21  UPDATE limpieza
22  SET Birth_date_formatted = CASE
23          WHEN SUBSTRING_INDEX(birth_date, '/', 1) > 12 THEN STR_TO_DATE(birth_date, '%d/%m/%Y')
24          ELSE STR_TO_DATE(birth_date, '%m/%d/%Y')
25      END;
```

La segunda parte de la consulta es esencial para la estructuración de nuestra base de datos. Aquí, utilizo la instrucción ALTER TABLE para agregar una nueva columna llamada Birth_date_formatted a la tabla limpieza.

- **ADD COLUMN**: Esto permite almacenar las fechas formateadas en un nuevo campo, lo cual es crucial para mantener la integridad de los datos y facilitar futuras consultas o análisis.

Finalmente, utilizo la instrucción UPDATE para llenar la columna recién creada con los valores convertidos.

- **SET**: Aquí se aplica nuevamente la lógica de CASE para determinar el formato correcto y actualizar la columna Birth_date_formatted con los valores correspondientes.
- La utilización de la misma lógica de formateo asegura que todos los registros en la nueva columna sean consistentes y correctos, mejorando la calidad de los datos en la base de datos.

Así quedaría antes y después de la consulta:

```
29 •  SELECT birth_date, birth_date_formatted FROM limpieza
30    LIMIT 5;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content

birth_date	birth_date_formatted
06/04/1991	1991-06-04
6/29/1984	1984-06-29
7/29/1989	1989-07-29
9/14/1982	1982-09-14
04/11/1994	1994-04-11

'LIMIT' se utiliza para limitar la consulta, en este caso para mostrarme los primeros 5 registros.

En cuanto a la columna 'Start_date' hice exactamente lo mismo, ya que presentaba las mismas inconsistencias:

```
43 •  UPDATE limpieza
44    SET star_date_formatted = CASE
45            WHEN SUBSTRING_INDEX(star_date, '/', 1) > 12 THEN STR_TO_DATE(star_date, '%d/%m/%Y')
46            ELSE STR_TO_DATE(star_date, '%m/%d/%Y')
47        END;
```

Quedando antes y después de la siguiente manera:

```
51 •    SELECT start_date, start_date_formatted FROM limpieza
52      LIMIT 5;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

start_date	start_date_formatted
1/20/2002	2002-01-20
04/08/2019	2019-04-08
10/12/2010	2010-10-12
04/10/2005	2005-04-10
9/29/2010	2010-09-29

Esto es fundamental, ya que trabajar con formatos inconsistentes puede llevar a errores en el análisis de datos. Al asegurarnos de que todas las fechas estén en el mismo formato, se puede llevar a cabo un análisis más eficaz y preciso.

Otro desafio importante fue la conversión de fechas en un formato de texto a un formato de fecha que SQL pudiera reconocer. Inicialmente, los registros de ‘finish_date’ estaban en el formato 2029-10-29 06:09:38 UTC. Para solucionarlo, utilice la función STR_TO_DATE, que me permitió convertir estos valores de texto en un formato de fecha estándar:

```
56 •    UPDATE limpieza
57      SET finish_date = STR_TO_DATE(finish_date, '%Y-%m-%d %H:%i:%s UTC')
58      WHERE finish_date IS NOT NULL AND finish_date != '';
```

Sin embargo, encontré errores debido a valores vacíos ya que esos registros hacen referencias a empleados que todavía siguen vigentes en la empresa. Ajustando la consulta para manejar estos casos, pude asegurarme de que los registros vacíos se convirtieran en NULL.

3. Separación de Fecha y Hora

Con las fechas correctamente formateadas, decidí dividir la columna ‘finish_date’ en dos: una para la fecha y otra para la hora. Esto mejoro la legibilidad y facilito análisis posteriores. La consulta fue:

```sql
ALTER TABLE limpieza
ADD COLUMN finish_date_only DATE,
ADD COLUMN finish_time_only TIME;

UPDATE limpieza
SET finish_date_only = IF(finish_date = '', NULL, DATE(finish_date)),
    finish_time_only = IF(finish_date = '', NULL, TIME(finish_date));
```

Esta separación permite realizar análisis mas específicos, como calcular la duración de empleo de manera mas precisa.

Quedando finalmente así:

```sql
SELECT finish_date, finish_date_only, finish_time_only FROM limpieza
LIMIT 5;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

finish_date	finish_date_only	finish_time_only
	NULL	NULL
	NULL	NULL
	NULL	NULL
	NULL	NULL
2029-10-29 06:09:38	2029-10-29	06:09:38

4. Manejo de Salarios en formato incorrect

Adicionalmente, la columna de salario presentaba datos en un formato que incluía símbolo de dólar y comas, lo que complica su análisis. Utilice una consulta para limpiar estos datos:

```sql
UPDATE limpieza
SET salary = CAST(REPLACE(REPLACE(salary, '$', ''), ',', '') AS DECIMAL(10, 2))
WHERE salary IS NOT NULL;
```

Esta limpieza fue crucial para asegurar que los salarios se pudieran utilizar en cálculos y reportes sin errores. Quite el símbolo de dólar y quite la coma. Además de eso, deje la columna ‘salary’ en formato DECIMAL ya que la misma estaba en formato texto.

5. Uso de operadores de comparación y funciones SQL avanzadas

Aparte de la limpieza de datos, también implemente operadores de comparación y funciones avanzadas. Utilice el operador INTERVAL.

La función INTERVAL se utiliza para manejar intervalos de fecha y hora sumando y restando intervalos como "3 días", "5 horas", "45 minutos".

```
113 •   SELECT
114         name,
115         last_name,
116         start_date_formatted + INTERVAL 2 DAY AS add_2days,
117         start_date_formatted - INTERVAL 3 DAY AS minus_3days,
118         finish_time_only + INTERVAL 4 HOUR AS add_4hours,
119         finish_date_only - INTERVAL 30 MINUTE AS minus_30mins
120     FROM limpieza
121     LIMIT 5;
122
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

name	last_name	add_2days	minus_3days	add_4hours	minus_30mins
Kimmy	Walczynski	2002-01-22	2002-01-17	NULL	NULL
Ignatius	Springett	2019-04-10	2019-04-05	NULL	NULL
Corbie	Bittlestone	2010-10-14	2010-10-09	NULL	NULL
Baxy	Matton	2005-04-12	2005-04-07	NULL	NULL
Terrell	Suff	2010-10-01	2010-09-26	10:09:38	2029-10-28 23:30:00

Por ejemplo, la utilice para calcular la antigüedad de los empleados.

```
107 •   SELECT name, last_name, TIMESTAMPDIFF(YEAR, start_date_formatted, CURRENT_DATE()) AS antigüedad
108     FROM limpieza
109     WHERE start_date_formatted <= CURRENT_DATE() - INTERVAL 1 YEAR
110     LIMIT 3;
111
```

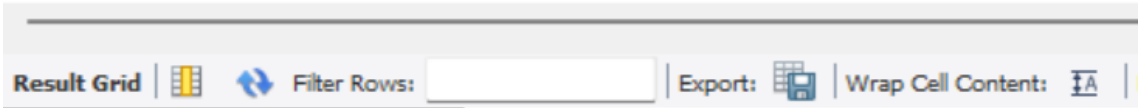
Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

name	last_name	antigü edad
Kimmy	Walczynski	22
Ignatius	Springett	5
Corbie	Bittlestone	13

TIMESTAMPDIFF(YEAR, start_date, CURRENT_DATE()) AS antigüedad: Calcula la diferencia en años entre la fecha de start_date y la fecha actual, y la renombra como antigüedad.

También explore el uso de la función EXTRACT (Extrae un componente especifico como año, mes, día, hora o minuto), que permite obtener partes especificas de una fecha. Esto es particularmente útil para análisis de rendimiento basados en años o meses:

```sql
SELECT
    EXTRACT(YEAR FROM start_date_formatted) AS año_inicio,
    COUNT(*) AS cantidad_empleados
FROM limpieza
GROUP BY año_inicio
HAVING COUNT(*) > 10
```



año_inicio	cantidad_empleados
2002	434
2019	408
2010	423
2005	432
2018	448

En esta consulta EXTRACT(YEAR FROM start_date_formatted) AS año_inicio extra el año de la columna start_date_formatted y lo renombra como año_inicio, COUNT(*) as cantidad_empleados cuenta el numero total de empleados para cada año, GROUP BY año_inicio agrupa los resultados por el año extraído y HAVING COUNT(*) > 10 filtra los grupos para mostrar solo aquellos años con mas de 10 empleados.

Conclusión

En el proceso de trabajar con la base de datos de empleados, he tenido la oportunidad de enfrentar diversos desafíos que me han permitido profundizar mis conocimientos en SQL y demostrar mis habilidades en la manipulación y limpieza de datos. A lo largo de este artículo, hemos explorado no solo cómo formatear fechas, sino también cómo abordar la calidad de los datos desde un enfoque integral.

Desde la estandarización de los nombres de las columnas hasta la conversión de formatos de fecha, cada paso ha sido fundamental para garantizar que la información se presente de manera consistente y efectiva. He aprendido que, más allá de realizar consultas básicas, es crucial entender la lógica detrás de cada transformación y cómo estas pueden impactar el análisis posterior. Por ejemplo, el uso de

CASE para identificar y convertir formatos de fecha no solo optimiza la limpieza de datos, sino que también prepara el terreno para análisis más complejos y significativos en el futuro.

Asimismo, trabajar con columnas que contienen valores nulos y vacíos ha sido un recordatorio constante de que, en el análisis de datos, la calidad es más importante que la cantidad. Aprender a manejar estos casos con elegancia, asegurando que la integridad de los datos se mantenga, es un signo de un analista consciente y profesional.

Además, he integrado conceptos más avanzados, como operadores de comparación y funciones como INTERVAL y EXTRACT, que han enriquecido aún más mi entendimiento de SQL. Estos elementos son cruciales para realizar análisis más sofisticados, permitiendo insights que pueden informar decisiones estratégicas en un entorno empresarial.

Finalmente, la capacidad de comunicar efectivamente lo aprendido a través de un artículo como este es esencial. La documentación clara y convincente de mis procesos no solo demuestra mis habilidades técnicas, sino que también muestra mi compromiso con la mejora continua y el aprendizaje en el ámbito de los datos. Cada desafío superado y cada consulta escrita son un paso más hacia la maestría en SQL, y estoy emocionado por aplicar todo este conocimiento en futuros proyectos y análisis.

A medida que continuo mi camino en el análisis de datos, sé que la práctica constante y la disposición para enfrentar nuevos desafíos serán clave para mi crecimiento profesional. Espero que este artículo no solo sea un reflejo de mis habilidades actuales, sino también un indicativo de mi ambición y determinación para seguir aprendiendo y evolucionando en el fascinante mundo de los datos.