Documentación del Proceso ETL y Análisis de Tech Store.

Documentación clara y detallada que enumera cada ejercicio del proceso ETL y el análisis realizado para este desafío 1.

Creado por Francisco Javier Gonzalez GS201626

1. Extracción de Datos

1.1 Objetivo:

Obtener datos de las fuentes proporcionadas (archivos CSV) para cargarlos en la base de datos SQL Server.

1.2 Archivos Utilizados:

- clientes.csv
- productos.csv
- ventas.csv

1.3 Pasos:

a) Verificación previa de los archivos CSV:

- Revisé los archivos para asegurarse de que tuvieran encabezados claros y datos consistentes.
- Verifiqué que los archivos no contenían líneas vacías o caracteres especiales que pudieran causar errores.

b) Creación de la base de datos:

Se creó una base de datos llamada `TechStore`:

```
sql
CREATE DATABASE TechStore;
GO
```

c) Creación de tablas en SQL Server:

Logré definir las estructuras de las tablas para `Clientes`, `Productos`, y `Ventas`.

```sql
-- Crear la tabla Clientes
CREATE TABLE Clientes (
id_cliente INT PRIMARY KEY,
nombre NVARCHAR(100),
ciudad NVARCHAR(100)
);

-- Crear la tabla Productos
CREATE TABLE Productos (
id_producto INT PRIMARY KEY,
nombre NVARCHAR(100),
precio DECIMAL(10, 2)
);

-- Crear la tabla Ventas
CREATE TABLE Ventas (
id_venta INT PRIMARY KEY,
id_cliente INT,
id_producto INT,cantidad INT,
FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente),
FOREIGN KEY (id_producto) REFERENCES Productos(id_producto)
);
```

d) Carga inicial de datos con BULK INSERT:

Inserté los datos desde los archivos CSV en las tablas correspondientes usando el comando BULK INSERT. Ejemplo:

```sql
BULK INSERT Clientes
FROM ' ‘C:\TechStore\clientes.csv
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\r\n',
    FIRSTROW = 2
);

BULK INSERT Productos
FROM 'C:\ruta\a\productos.csv'
WITH (
    FIELDTERMINATOR = ',',
```

```
    ROWTERMINATOR = '\n',
    FIRSTROW = 2
);

BULK INSERT Ventas
FROM 'C:\ruta\a\ventas.csv'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2
);
```

2. Transformación de Datos

2.1 Objetivo:

Limpiar, validar y transformar los datos para garantizar su integridad y prepararlos para el análisis.

2.2 Pasos:

a) Validación de datos:

Verifiqué la existencia de valores nulos o inconsistentes:

```
 sql
SELECT * FROM Clientes WHERE nombre IS NULL OR ciudad IS NULL;
```

Al igual que verifiqué que no hubieran ventas con referencias inválidas:

```
 sql
SELECT *
FROM Ventas
WHERE id_cliente NOT IN (SELECT id_cliente FROM Clientes)
   OR id_producto NOT IN (SELECT id_producto FROM Productos);
```

b) Limpieza y conversión de tipos de datos:

Ajusté tipos de datos para que fueran consistentes con el uso previsto. Por ejemplo:

```sql
ALTER TABLE Ventas
ALTER COLUMN cantidad INT;

ALTER TABLE Clientes
ALTER COLUMN nombre nvarch(50);
```

c) Cálculos adicionales:

Calculé el monto total de cada venta:

```sql
SELECT v.id_venta, (v.cantidad * p.precio) AS monto_total
FROM Ventas v
JOIN Productos p ON v.id_producto = p.id_producto;
```

Generé el código de país a partir del id_cliente:

```sql
SELECT id_cliente, LEFT(id_cliente, 2) AS codigo_pais
FROM Clientes;
```

d) Segmentación de clientes:

Clasifiqué a los clientes en "Frecuentes" y "Ocasionales" según el número de compras:

```sql
SELECT c.id_cliente, COUNT(v.id_venta) AS num_compras,
       CASE
           WHEN COUNT(v.id_venta) >= 5 THEN 'Frecuente'
           ELSE 'Ocasional'
       END AS categoria
FROM Ventas v
JOIN Clientes c ON v.id_cliente = c.id_cliente
GROUP BY c.id_cliente;
```

3. Carga de Datos Procesados

3.1 Objetivo:

Almacenar los datos transformados en tablas finales y exportar información segmentada.

3.2 Pasos:

a) Tablas segmentadas:

Creé tablas para almacenar clientes segmentados:

```
sql
CREATE TABLE Clientes_Frecuentes AS
SELECT * FROM Clientes WHERE id_cliente IN (
    SELECT id_cliente
    FROM Ventas
    GROUP BY id_cliente
    HAVING COUNT(id_venta) >= 5
);
```

b) Exportación a Excel:

Los datos segmentados se exportaron a archivos de Excel (`Clientes_Frecuentes.xlsx` y `Clientes_Ocasionales.xlsx`) utilizando la herramienta de exportación de SQL Server Management Studio.

4. Análisis y Consultas SQL

4.1 Objetivo:

Proveer información relevante para decisiones estratégicas basadas en los datos procesados.

4.2 Consultas realizadas:

a) Clientes frecuentes y su monto total de compras:

```
 sql
SELECT c.id_cliente, c.nombre, SUM(v.cantidad * p.precio) AS
total_compras
FROM Ventas v
JOIN Clientes c ON v.id_cliente = c.id_cliente
JOIN Productos p ON v.id_producto = p.id_producto
WHERE c.id_cliente IN (
    SELECT id_cliente
    FROM Ventas
    GROUP BY id_cliente
    HAVING COUNT(id_venta) >= 5
)
GROUP BY c.id_cliente, c.nombre;
```

b) Productos más vendidos:

```
 sql
SELECT p.nombre, SUM(v.cantidad) AS total_vendido
FROM Ventas v
JOIN Productos p ON v.id_producto = p.id_producto
GROUP BY p.nombre
ORDER BY total_vendido DESC;
```

c) Ventas por región (basada en el código de país):

```
 sql
SELECT LEFT(c.id_cliente, 2) AS region, SUM(v.cantidad * p.precio)
AS total_ventas
FROM Ventas v
JOIN Clientes c ON v.id_cliente = c.id_cliente
JOIN Productos p ON v.id_producto = p.id_producto GROUP
BY LEFT(c.id_cliente, 2);
```

5. Uso de Git y Control de Cambios

5.1 Objetivo:

Mantener un historial de los scripts y cambios realizados.

5.2 Pasos:

a) Repositorio inicial:

Creé un repositorio Git para gestionar los scripts SQL y archivos relacionados:
bash git
init

b) Control de versiones:

Realicé commits frecuentes con mensajes claros para cada cambio:
bash
git add scripts.sql
git commit -m "Creación de tablas y carga de datos inicial"

Repository: https://github.com/francgonzalez1/TechStore_ETL.git

6. Documentación:

Guardé este documento junto con los scripts y el archivo README.md explicando cada paso.