PROFESORES:

Felipe Gonzalez Casabianca (Sección 3)

INTEGRANTES:

Maria Camila Parra Díaz (201819464)
Esteban Emmanuel Ortiz Morales (201913613)
Sergio Julian Zona Moreno (201914936)

Introducción

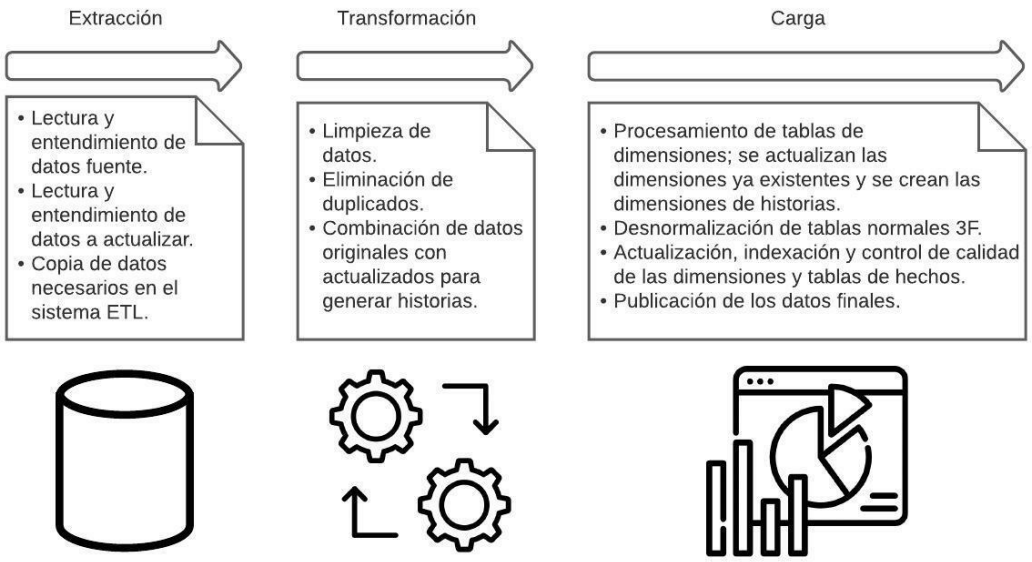
En el presente documento desarrollaremos el laboratorio 5 del curso de Inteligencia de Negocios. El propósito central es dar solución a la segunda parte del caso de estudio presentado por World Wide Importers (WWI), según lo adquirido en las clases de modelado multidimensional. Se utilizarán como herramientas Spoon y Python (con ejecución en un cuadernillo de Jupyter) para el modelado del proceso ETL y la creación de las historias en las dimensiones StockItem, Customer, y en la tabla de hechos FactOrder.

Aclaraciones preliminares

- Se recomienda al calificador/lector de este documento visualizarlo con un Zoom de 160% para evitar forzar la vista. Además, este hecho permite ver con nitidez los gráficos y las tablas presentadas.
- Al finalizar este documento se encuentran los anexos que redirigen a un repositorio con los resultados obtenidos e implementaciones efectuadas durante el laboratorio.

Diagrama de alto nivel del proceso ETL

A continuación, se presenta el diagrama realizado acerca del proceso ETL:



A diferencia del primer modelo ETL en el laboratorio 4, este incorpora en cada una de las fases el uso de historias. En la extracción, se leen los datos nuevos y viejos y se contrastan. En la transformación, se combinan los datos nuevos con los originales para generar versionamiento

(adición de atributos como fecha y número de versión). Por último, en la carga, se dejan constantes los valores que no fueron modificados y se agregan las dimensiones que manejan las historias.

Ventajas y desventajas del manejo de historias

Ventajas	Desventajas
-Se mantiene un registro de todos los estados del conjunto de datos. Esto implica conocer con claridad cómo fueron los mismos en determinado momento del tiempo. -Permite conocer las modificaciones en el conjunto de datos entre ciertos lapsos específicos, conociendo qué dimensiones fueron modificadas (además de las tablas de hechos) y de qué manera. -Al realizar Querys y llamadas al conjunto de datos, estos pueden ser muy rápidos y traer gran cantidad de registros en un tiempo razonable.	-Redundancia exagerada en el conjunto de datos, esto implica que se necesita de un hardware de almacenamiento bastante robusto. -Al transformar los datos, es necesario efectuar Querys de alto nivel de complejidad, que realicen Joins entre los datos antiguos y los datos actualizados. Adicionalmente se debe tener en cuenta la creación de las dimensiones de historias y ajustar estos valores en las tablas de hechos. -La actualización de las tablas de hechos y dimensiones es un proceso demorado que requiere de un nivel de procesamiento computacional y tiempo adecuado.

Ventajas y desventajas de los tipos implementados (comparativa)

En el laboratorio utilizamos los tipos I, II y III. La comparativa se presenta a continuación:

Tipo	Ventajas	Desventajas
Tipo I	- Dado que sobrescribe las versiones antiguas, no almacena grandes cantidades de datos y la redundancia es menor. A nivel de hardware de almacenamiento no se requiere de una gran capacidad comparado con los otros tipos.	- No almacena versiones antiguas del conjunto de datos, por lo que es imposible estados antiguos del conjunto de datos.
Tipo II	- Permite tener un historial de versiones entre los datos, por lo que existe una mayor redundancia de los mismos y se puede acceder a estados antiguos de los datos. - Al tener un historial de los datos, se pueden determinar las modificaciones entre dos	- A nivel de hardware de almacenamiento, se utiliza más espacio por la redundancia de los datos. - Se necesitan más recursos de procesamiento y los tiempos de actualización son mayores.

	tipos de versiones efectuando una comparativa.	
Tipo III	- Crea un nuevo atributo (columna en una dimensión o tabla de hechos), que contiene la modificación nueva efectuada en el conjunto de datos. Además, genera un atributo con la fecha en que fue realizada la última modificación al registro en cuestión. - Genera un versionamiento que no requiere de tanta capacidad de almacenamiento. Este tipo de forma de implementar historial es particularmente útil cuando se quieren comparar versiones entre atributos y no de todo el registro.	- Solo permite almacenar un número limitado de versiones (generalmente respecto a la última modificación efectuada a un atributo). Esto implica que no se pueden acceder a estados muy antiguos del conjunto de datos.

Preprocesamiento

Para los datos nuevos se presentaron inconsistencias que debieron ser corregidas, estas inconsistencias fueron:

- Registros con valores “Unknown” en diversos registros de las tablas.
- Los valores decimales se encontraban con “,” y fueron cambiados a “.”.

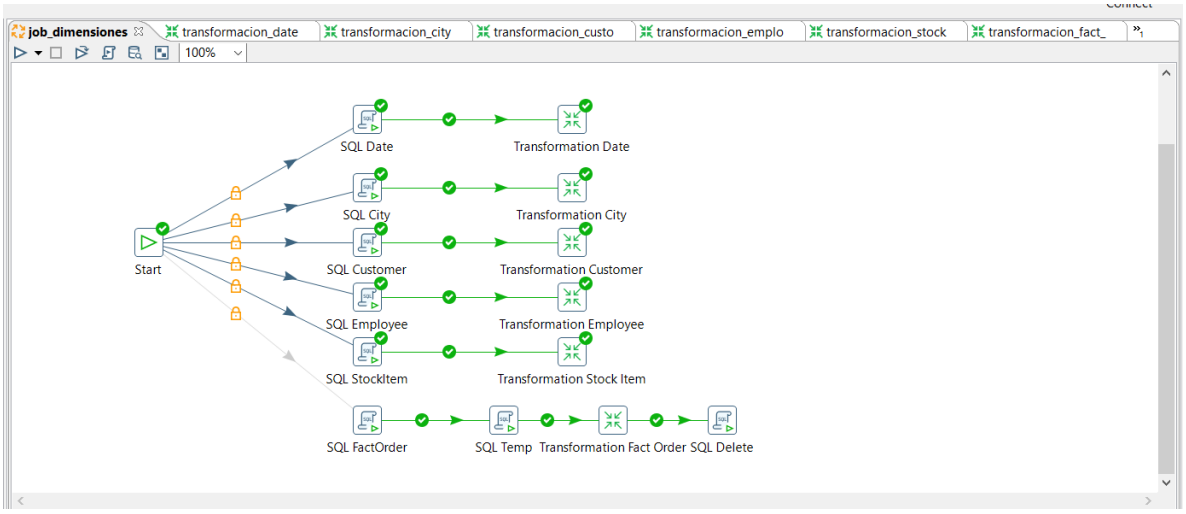
El cuadernillo de depuración puede ser observado en el repositorio de anexos.

Documentación

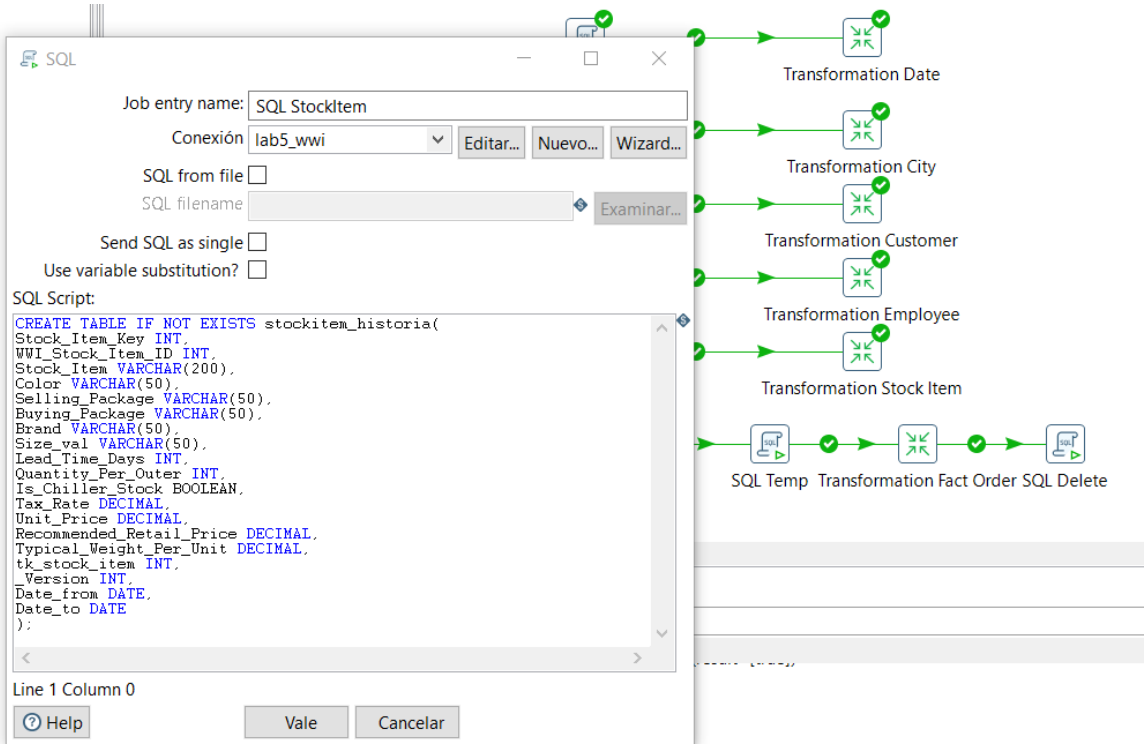
Paralelo a la implementación de nuestras herramientas ETL, utilizamos una base de datos Postgres SQL que almacenará el conjunto de datos transformado. A continuación, se procede a detallar los procesos implementados:

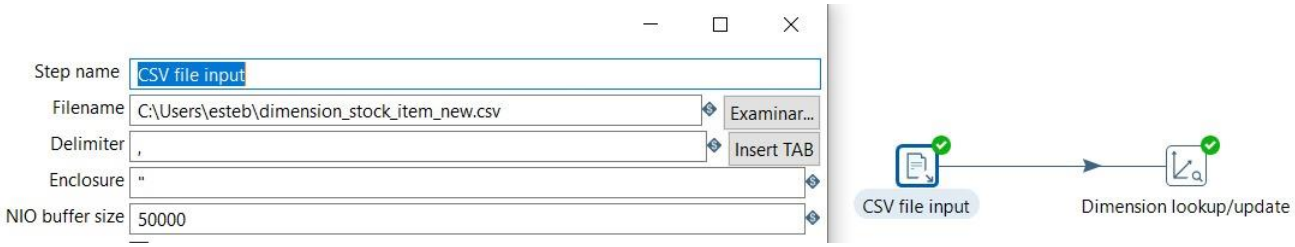
Documentación del proceso y transformaciones en Spoon

Foro del proceso final logrado desde alto nivel, a continuación, se irán mostrando imágenes que muestren la implementación de las transformaciones en casa uno de sus componentes:

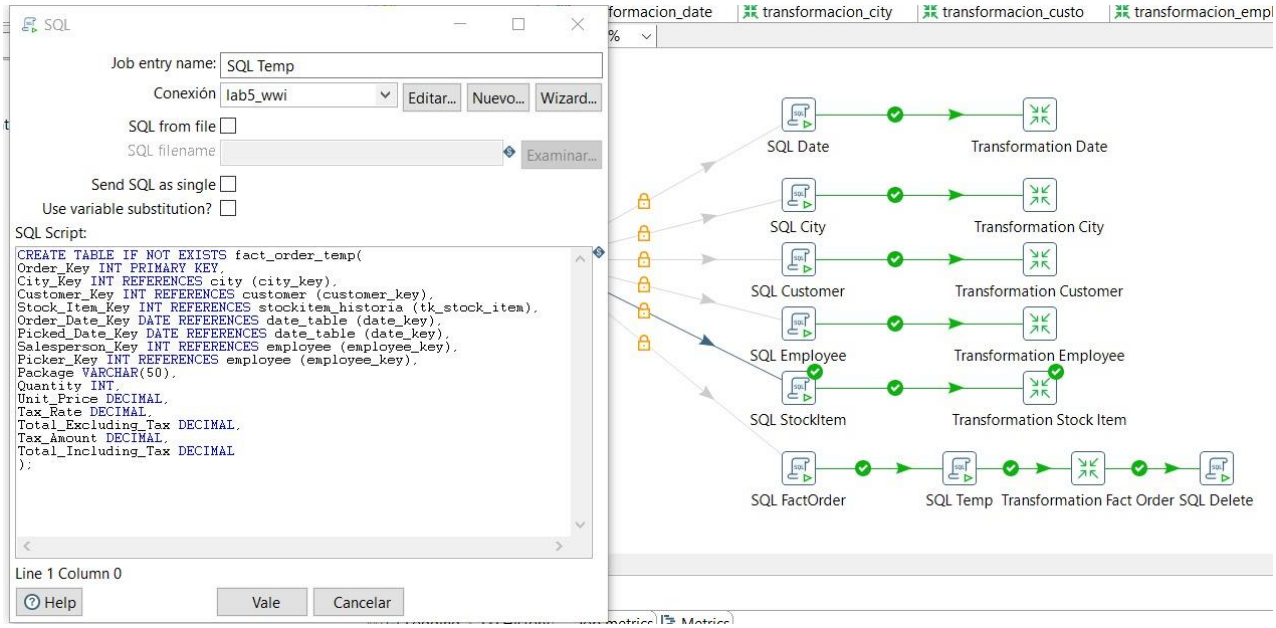


Se crea mediante sentencia SQL la tabla de historia de stockitem, luego se le aplica la respectiva transformación que trae los datos del csv stock_item_new y los inserta

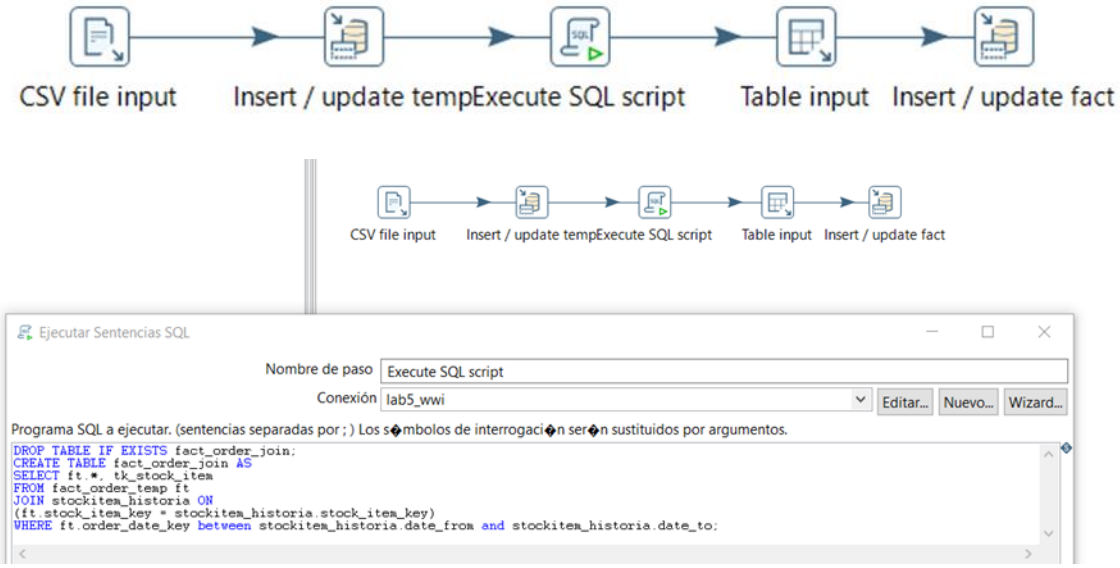


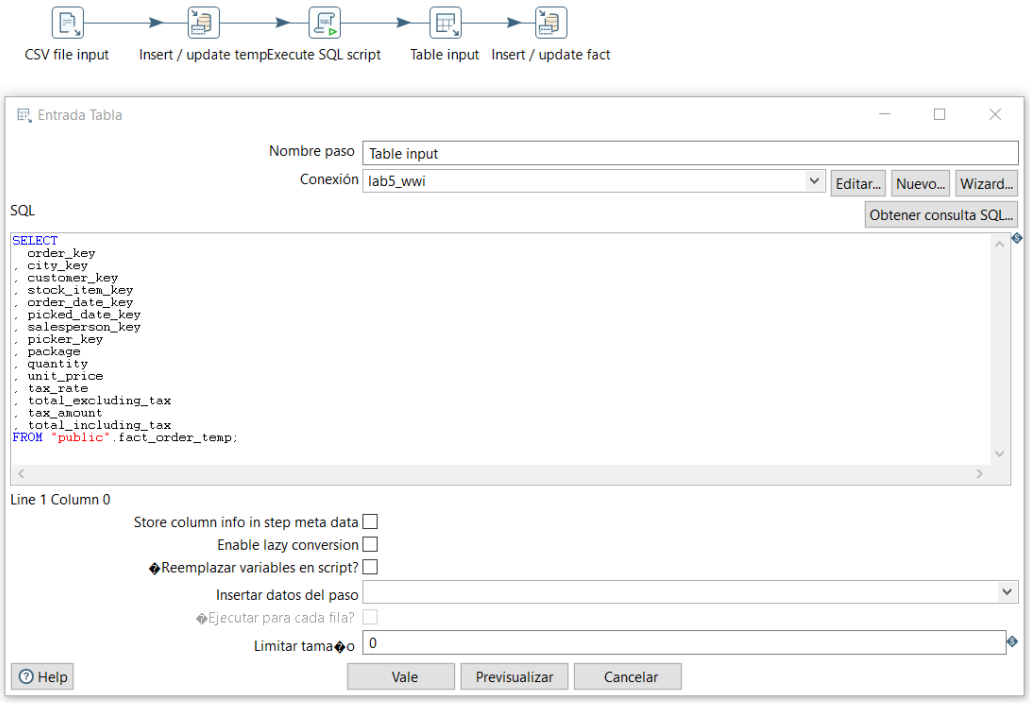


Se crea mediante sentencia SQL la tabla de historia de fact_order_temp, tabla que servirá para hacer los joins necesarios con el objetivo de actualizar la tabla de hechos a partir de la nueva información brindada por el CSV, nótese que al fact_order_temp se elimina en SQL Delete dado que solo es necesaria para los join.



Dentro de la transformación fact table, se detalla un poco más el proceso realizado, en donde se leen los datos del csv, se insertan en la tabla temporal, se ejecuta el script sql que hace un join en los ids de Stock_Item respetando los rangos de fecha inicial y final de cada uno de los registros, finalmente los valores resultados son seleccionados y actualizados en la tabla de fact_order.





Una vez hechos todos estos pasos deberíamos poder hacer las consultas a nuestra base de datos, a continuación, se mostrarán los resultados de las sentencias en cada una de la columna de interés.

Documentación del proceso y transformaciones en Python (segunda herramienta seleccionada)

En el repositorio de anexos se pueden encontrar los cuadernillos que presentan la implementación efectuada, estos cuadernillos detallan con más claridad todo el proceso desarrollado.

Configuración con la base de datos:

```
# Configuración de la conexión con la base de datos

# se hacen las importaciones y configuraciones necesarias necesarias para trabajar con SQL
from sqlalchemy import create_engine

# Postgres username, password, and database name
POSTGRES_ADDRESS = 'localhost'
POSTGRES_PORT = '5432'
POSTGRES_USERNAME = 'postgres'
POSTGRES_PASSWORD = 'password'
POSTGRES_DBNAME = 'lab5_wwi'
# A long string that contains the necessary Postgres login information
postgres_str = ('postgresql://{username}:{password}@{ipaddress}:{port}/{dbname}'
.format(username=POSTGRES_USERNAME,
password=POSTGRES_PASSWORD,
ipaddress=POSTGRES_ADDRESS,
port=POSTGRES_PORT,
dbname=POSTGRES_DBNAME))
# Create the connection
cnx = create_engine(postgres_str)
```

Stock Item:

En esta dimensión se implementa un manejo de historias tipo II. Primero depuramos los datos:

```
# Depuracion de StockItem con los valores a actualizar

# se elimina la primera fila de los nuevos datos porque todos sus valores son nulos o desconocidos
df_stock_item_new = df_stock_item_new[df_stock_item_new.Stock_Item != 'Unknown']
# se elimina la columna brand, dado que los datos originales no la tienen
df_stock_item_new.drop(['Brand'], axis = 1, inplace=True)
df_stock_item_new['Tax_Rate'] = [x.replace(',','.') for x in df_stock_item_new['Tax_Rate']]
df_stock_item_new['Unit_Price'] = [x.replace(',','.') for x in df_stock_item_new['Unit_Price']]
df_stock_item_new['Recommended_Retail_Price'] = [x.replace(',','.') for x in df_stock_item_new['Recommended_Retail_Price']]
df_stock_item_new['Typical_Weight_Per_Unit'] = [x.replace(',','.') for x in df_stock_item_new['Typical_Weight_Per_Unit']]
df_stock_item_new["Color"].fillna("NAN", inplace=True)
df_stock_item_new["Size_val"].fillna("NAN", inplace=True)

df_stock_item_new.head(5)
```

Se agrega un versionamiento de los valores actuales:

```
import datetime
```

```
stockitem_historia=stockitem_df

#Añadimos un versionamiento a los valoes actuales
stockitem_historia["Version"]=1

# añadimos fechas a los valores actuales
stockitem_historia["date_from"]= datetime.datetime(1900, 1, 1)
stockitem_historia["date_to"]= datetime.datetime(2199, 12, 31)

stockitem_historia.head(5)
```

Si se actualiza un registro, se cambia el versionamiento y la fecha:

```
# ahora lo que se necesita hacer es crear un algoritmo que garantice que con el ingreso de los nuevos datos
# se pueda modelar una Slowly Changing Dimension de tipo 2

# el algoritmo es el siguiente:
# se recorre todo el dataframe de los datos nuevos y se pone atención en el valor de stock_item_key
# si dicho valor existe en la tabla original de datos y alguno de los atributos iniciales ha cambiado, se procederá a hacer la inserción d
# la versión del nuevo registró será igual al número de la versión más reciente que contiene dicho id más 1
# asimismo, a la versión más reciente se le cambiará la fecha final por la fecha actual
# y a la nueva versión se le pondrá como fecha inicial la actual y como fecha final 2199-12-31

for row in df_stock_item_new.iterrows():
    stock_item_key = row[1]["Stock_Item_Key"]

    version = 0
    # ahora se recorre el antiguo dataframe
    indice = 0
    indice_real = 0
    usar_indice_real = False
    for row2 in stockitem_historia.iterrows():
        key = row2[1]["stock_item_key"]
        if stock_item_key == key:
            ver = row2[1]["Version"] # version
            if(ver > version):
                version = ver
                indice_real = indice
            usar_indice_real = True
        indice+=1
```

```
# hay que detectar si hubo algún cambio respecto a la version final
if( usar_indice_real and
    (row[1]["Stock_Item"] != stockitem_historia.loc[indice_real]["stock_item"] or
    row[1]["Color"] != stockitem_historia.loc[indice_real]["color"] or
    row[1]["Selling_Package"] != stockitem_historia.loc[indice_real]["selling_package"] or
    row[1]["Buying_Package"] != stockitem_historia.loc[indice_real]["buying_package"] or
    row[1]["Size_val"] != stockitem_historia.loc[indice_real]["size_val"] or
    row[1]["Lead_Time_Days"] != stockitem_historia.loc[indice_real]["lead_time_days"] or
    row[1]["Quantity_Per_Outer"] != stockitem_historia.loc[indice_real]["quantity_per_outer"] or
    row[1]["Is_Chiller_Stock"] != stockitem_historia.loc[indice_real]["is_chiller_stock"] or
    row[1]["Tax_Rate"] != stockitem_historia.loc[indice_real]["tax_rate"] or
    row[1]["Unit_Price"] != stockitem_historia.loc[indice_real]["unit_price"] or
    row[1]["Recommended_Retail_Price"] != stockitem_historia.loc[indice_real]["recommended_retail_price"] or
    row[1]["Typical_Weight_Per_Unit"] != stockitem_historia.loc[indice_real]["typical_weight_per_unit"] ) ):

        dt = datetime.datetime.now()
        stockitem_historia.loc[indice_real,['date_to']] = datetime.datetime(dt.year, dt.month, dt.day)
        row_a_agregar = stockitem_historia.loc[indice_real]
        row_a_agregar["Version"] = version + 1
        dt = datetime.datetime.now()

        row_a_agregar["stock_item"] = row[1]["Stock_Item"]
        row_a_agregar["color"] = row[1]["Color"]
        row_a_agregar["selling_package"] = row[1]["Selling_Package"]
        row_a_agregar["buying_package"] = row[1]["Buying_Package"]
        row_a_agregar["size_val"] = row[1]["Size_val"]
        row_a_agregar["lead_time_days"] = row[1]["Lead_Time_Days"]
        row_a_agregar["quantity_per_outer"] = row[1]["Quantity_Per_Outer"]
        row_a_agregar["is_chiller_stock"] = row[1]["Is_Chiller_Stock"]
        row_a_agregar["tax_rate"] = row[1]["Tax_Rate"]
        row_a_agregar["unit_price"] = row[1]["Unit_Price"]
        row_a_agregar["recommended_retail_Price"] = row[1]["Recommended_Retail_Price"]
        row_a_agregar["typical_Weight_Per_Unit"] = row[1]["Typical_Weight_Per_Unit"]

        row_a_agregar["date_from"] = datetime.datetime(dt.year, dt.month, dt.day)
        row_a_agregar["date_to"] = datetime.datetime(2199, 12, 31)
        stockitem_historia = stockitem_historia.append(row_a_agregar, ignore_index=True)
```

Agregamos la llave subrogada:

```
# como ahora solo queda ordenar el dataframe en función del stock item key para que se pueda notar el cambio de los re
stockitem_historia = stockitem_historia.sort_values(['stock_item_key', 'date_from'])

# reset indexes
stockitem_historia = stockitem_historia.reset_index(drop=True)

# ahora se le coloca una llave subrogada que identifique a cada uno de los valores de la tabla
stockitem_historia["tk_stock_item"] = stockitem_historia.index + 1

# shift column 'tk_stock_item' to first position
first_column = stockitem_historia.pop('tk_stock_item')
stockitem_historia.insert(0, 'tk_stock_item', first_column)

stockitem_historia.head(20)
```

Fact Order:

Se cambia la referencia de la tabla de hechos para utilizar la llave truncada.

```
# Paso 1

# se eliminan los datos orignales de la tabla de hechos (Proceso realizado desde pgAdmin)
# se cambia la referencia de la tabla de hechos para que  ahora referencie a la llave truncada de la historia de stock_iter
def cambiar_date_time(x):
    return pd.to_datetime(x,format= '%Y-%m-%d' ).to_datetime64()


# carga de la tabla de los nuevos datos
df_fact_order_new =pd.read_csv('fact_order_new.csv', sep=',', encoding = 'latin-1', index_col=None)

# se cambiar el formato de la fecha para poder compararse
df_fact_order_new['picked_date_key'] = df_fact_order_new['picked_date_key'].apply(cambiar_date_time)

df_fact_order_new.head(5)
```

Se efectúa un join entre el ID de las órdenes en las tablas de historia y Stock Item dado un rango de fechas determinado.

```
# Paso 2

# se lee la historia
stockitem_historia=pd.read_sql_query('select * from stockitem_historia',cnx)

# se simula un join, en este caso sobre los ids, el resultado será que quedará la tabla de fact_order solamente con los va
df_empty = df_fact_order_new[0:0]
df_empty
indice = 0;
for row in df_fact_order_new.iterrows():
    date = row[1]["order_date_key"]
    date = pd.to_datetime(date, format='%Y-%m-%d')
    stock_item_key = row[1]["stock_item_key"]

    for row2 in stockitem_historia.iterrows():

        key = row2[1]["tk_stock_item"]
        stock_item_date_from  = row2[1]["date_from"].to_datetime64()
        stock_item_date_to = row2[1]["date_to"].to_datetime64()

        # llave igual y fecha entre las fechas de inicio y final del respectivo stock_item
        if stock_item_key == key and date >= stock_item_date_from and date <= stock_item_date_to:
            a_agregar = df_fact_order_new.iloc[indice]
            df_empty = df_empty.append(a_agregar, ignore_index=True)
    indice+= 1
```

Customer:

Para esta dimensión se nos solicitó implementar los tres tipos de historias.

Tipo I:

SDC Type 1: Sobreescribir. - implementation Customer

```
# carga de la tabla de los nuevos datos
df_customer_new =pd.read_csv('dimension_customer_new.csv', sep=',', encoding = 'latin-1', index_col=None)

# se carga la tabla con los datos originale de la base de datos
df_customer =pd.read_sql_query('select * from customer',cnx)

# se recorre la tabla original y si existen llaves iguales se sobreescribe
indice = 0
for row in df_customer_new.iterrows():
    key_new = row[1]["Customer_Key"]

    indice2 = 0
    for row2 in df_customer.iterrows():

        key = row2[1]["customer_key"]

        # llave igual
        if key_new == key:
            df_customer.iloc[indice] = df_customer_new.iloc[indice2]
        indice2+=1
    indice+= 1
```

Tipo II:

SDC Type 2: Se añade una nueva row. - implementation Customer

```
import datetime

# carga de la tabla de los nuevos datos
df_customer_new =pd.read_csv('dimension_customer_new.csv', sep=',', encoding = 'latin-1', index_col=None)

# se carga la tabla con los datos originale de la base de datos
df_customer =pd.read_sql_query('select * from customer',cnx)

#Añadimos un versionamiento a los valoes actuales
df_customer_historia = df_customer
df_customer_historia["Version"]=1

# añadimos fechas a los valores actuales
df_customer_historia["date_from"]= datetime.datetime(1900, 1, 1)
df_customer_historia["date_to"]= datetime.datetime(2199, 12, 31)
```

```
# se actualizan los valores con las nuevas versiones
for row in df_customer_new.iterrows():
    key_new = row[1]["Customer_Key"]

    version = 0
    # ahora se recorre el antiguo dataframe
    indice = 0
    indice_real = 0
    usar_indice_real = False
    for row2 in df_customer_historia.iterrows():
        key = row2[1]["customer_key"]
        if key_new == key:
            ver = row2[1]["Version"] # version
            if(ver > version):
                version = ver
                indice_real = indice
            usar_indice_real = True
        indice+=1

    # hay que detectar si hubo algún cambio respecto a la version final
    if( usar_indice_real and row[1]["Category"] != df_customer_historia.loc[indice_real]["category"] ):

            dt = datetime.datetime.now()
            df_customer_historia.loc[indice_real,['date_to']] = datetime.datetime(dt.year, dt.month, dt.day)
            row_a_agregar = df_customer_historia.loc[indice_real]
            row_a_agregar["Version"] = version + 1
            row_a_agregar["category"] = row[1]["Category"]
            dt = datetime.datetime.now()
            row_a_agregar["date_from"] = datetime.datetime(dt.year, dt.month, dt.day)
            row_a_agregar["date_to"] = datetime.datetime(2199, 12, 31)
            df_customer_historia = df_customer_historia.append(row_a_agregar, ignore_index=True)
```

```
# como ahora solo queda ordenar el dataframe en función del stock item key para que se pueda notar el cambio de los registros respecto a una SCl
df_customer_historia = df_customer_historia.sort_values(['customer_key', 'date_from'])

# reset indexes
df_customer_historia = df_customer_historia.reset_index(drop=True)

# ahora se le coloca una llave subrogada que identifique a cada uno de los valores de la tabla
df_customer_historia["tk_customer"] = df_customer_historia.index + 1

# shift column 'tk_stock_item' to first position
first_column = df_customer_historia.pop('tk_customer')
df_customer_historia.insert(0, 'tk_stock_item', first_column)

df_customer_historia.head(20)
```

Tipo III:

SDC Type 3: Se añade una nueva row. - implementation Customer

```
import datetime

# carga de la tabla de los nuevos datos
df_customer_new =pd.read_csv('dimension_customer_new.csv', sep=',', encoding = 'latin-1', index_col=None)

# se carga la tabla con los datos originale de la base de datos
df_customer =pd.read_sql_query('select * from customer',cnx)

#Añadimos un versionamiento a los valoes actuales
df_customer_historia = df_customer
df_customer_historia["previous_category"] = "NAN"
df_customer_historia["new_category"] = df_customer_historia["category"]

# añadimos fechas a los valores actuales
dt = datetime.datetime.now()

df_customer_historia['date_updated'] = datetime.datetime(dt.year, dt.month, dt.day)
```

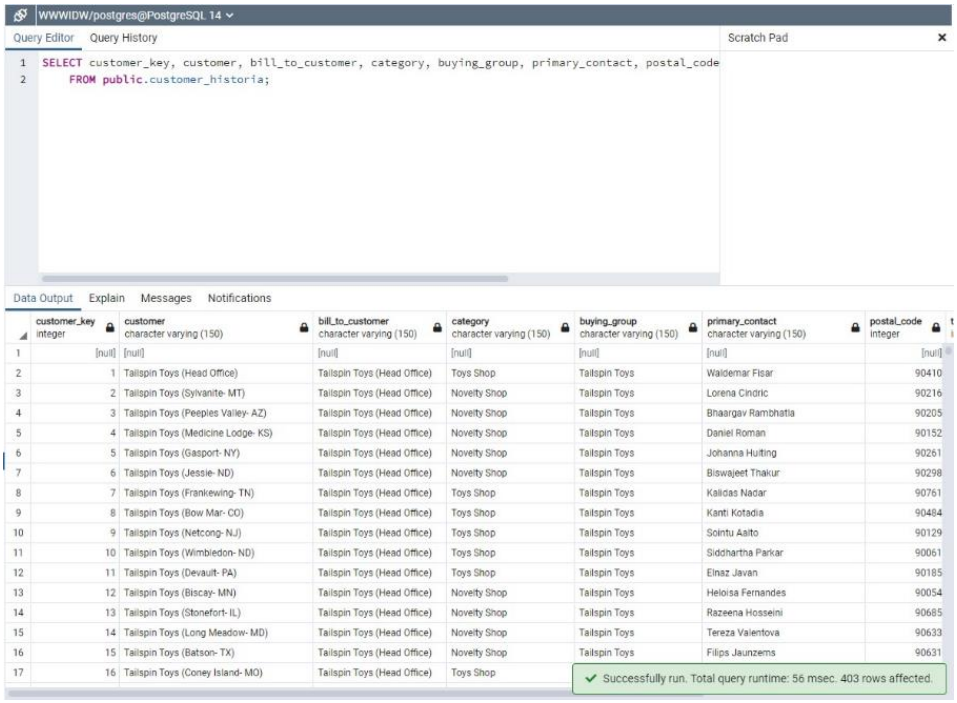
```
# se actualizan los valores con las nuevas versiones
for row in df_customer_new.iterrows():
    key_new = row[1]["Customer_Key"]
    # ahora se recorre el antiguo dataframe
    indice = 0
    indice_real = 0
    usar_indice_real = False
    for row2 in df_customer_historia.iterrows():
        key = row2[1]["customer_key"]
        if key_new == key:
            indice_real = indice
            usar_indice_real = True
        indice+=1

    # hay que detectar si hubo algún cambio respecto a la version final
    if( usar_indice_real and row[1]["Category"] != df_customer_historia.loc[indice_real]["new_category"] ):

        dt = datetime.datetime.now()
        df_customer_historia.loc[indice_real,['date_updated']] = datetime.datetime(dt.year, dt.month, dt.day)

        df_customer_historia.loc[indice_real, ['previous_category']] = df_customer_historia.loc[indice_real]["new_category"]
        df_customer_historia.loc[indice_real, ["new_category"]] = row[1]["Category"]

df_customer_historia = df_customer_historia.drop('category',1)
```

```
# como ahora solo queda ordenar el dataframe en función del stock item key para que se pueda notar el cambio de los registros respecto a una SCI
df_customer_historia = df_customer_historia.sort_values(['customer_key'])

# reset indexes
df_customer_historia = df_customer_historia.reset_index(drop=True)

df_customer_historia.head(20)
```

Resultados de implementación

Resultados en Spoon

Customer:



WWWIDW/postgres@PostgreSQL 14

Query Editor | Query History | Scratch Pad

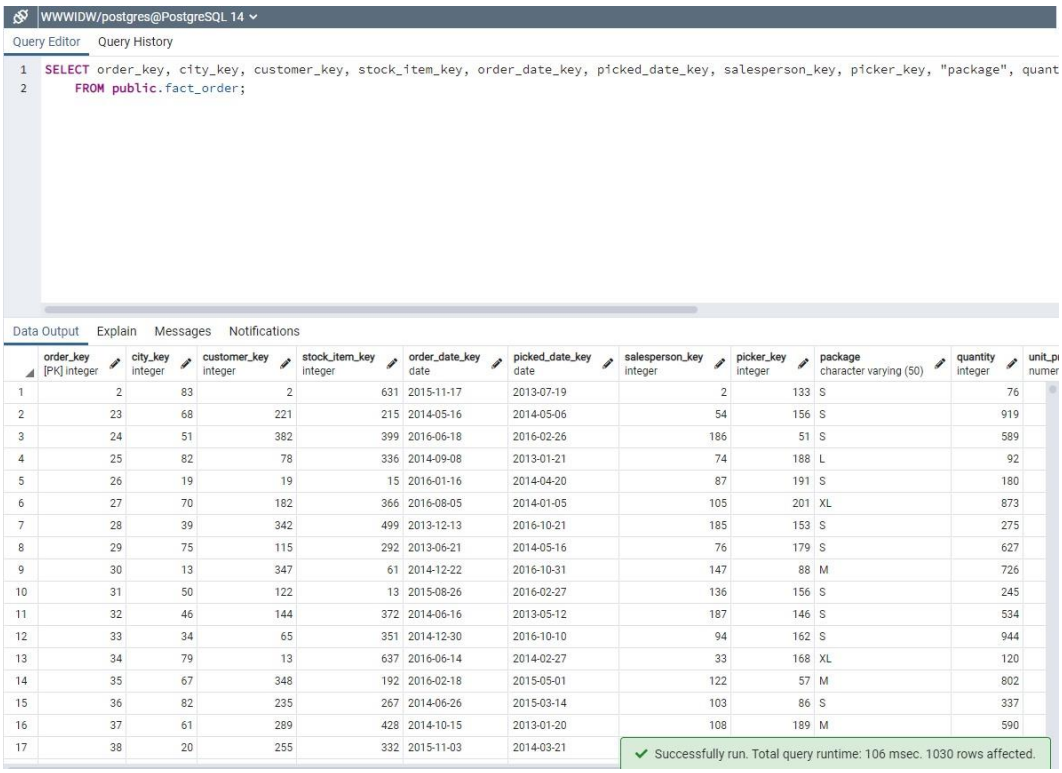
```
SELECT customer_key, customer, bill_to_customer, category, buying_group, primary_contact, postal_code
    FROM public.customer_historia;
```

Data Output | Explain | Messages | Notifications

	customer_key integer	customer character varying (150)	bill_to_customer character varying (150)	category character varying (150)	buying_group character varying (150)	primary_contact character varying (150)	postal_code integer
1	[null]	[null]	[null]	[null]	[null]	[null]	[null]
2	1	Tailspin Toys (Head Office)	Tailspin Toys (Head Office)	Toys Shop	Tailspin Toys	Waldemar Fisar	90410
3	2	Tailspin Toys (Sylvanite- MT)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Lorena Cindric	90216
4	3	Tailspin Toys (Peeples Valley- AZ)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Bhaargav Rambhatla	90205
5	4	Tailspin Toys (Medicine Lodge- KS)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Daniel Roman	90152
6	5	Tailspin Toys (Gasport- NY)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Johanna Huiting	90261
7	6	Tailspin Toys (Jessie- ND)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Biswajeet Thakur	90298
8	7	Tailspin Toys (Frankewing- TN)	Tailspin Toys (Head Office)	Toys Shop	Tailspin Toys	Kalidas Nadar	90761
9	8	Tailspin Toys (Bow Mar- CO)	Tailspin Toys (Head Office)	Toys Shop	Tailspin Toys	Kanti Kotadia	90484
10	9	Tailspin Toys (Netcong- NJ)	Tailspin Toys (Head Office)	Toys Shop	Tailspin Toys	Sointu Aalto	90129
11	10	Tailspin Toys (Wimbledon- ND)	Tailspin Toys (Head Office)	Toys Shop	Tailspin Toys	Siddhartha Parkar	90061
12	11	Tailspin Toys (Devault- PA)	Tailspin Toys (Head Office)	Toys Shop	Tailspin Toys	Elnaz Javan	90185
13	12	Tailspin Toys (Biscay- MN)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Heloisa Fernandes	90054
14	13	Tailspin Toys (Stonefort- IL)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Razeena Hosseini	90685
15	14	Tailspin Toys (Long Meadow- MD)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Tereza Valentova	90633
16	15	Tailspin Toys (Batson- TX)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Filips Jaunzems	90631
17	16	Tailspin Toys (Coney Island- MO)	Tailspin Toys (Head Office)	Toys Shop			

Successfully run. Total query runtime: 56 msec. 403 rows affected.

Fact Order:



WWWIDW/postgres@PostgreSQL 14

Query Editor | Query History

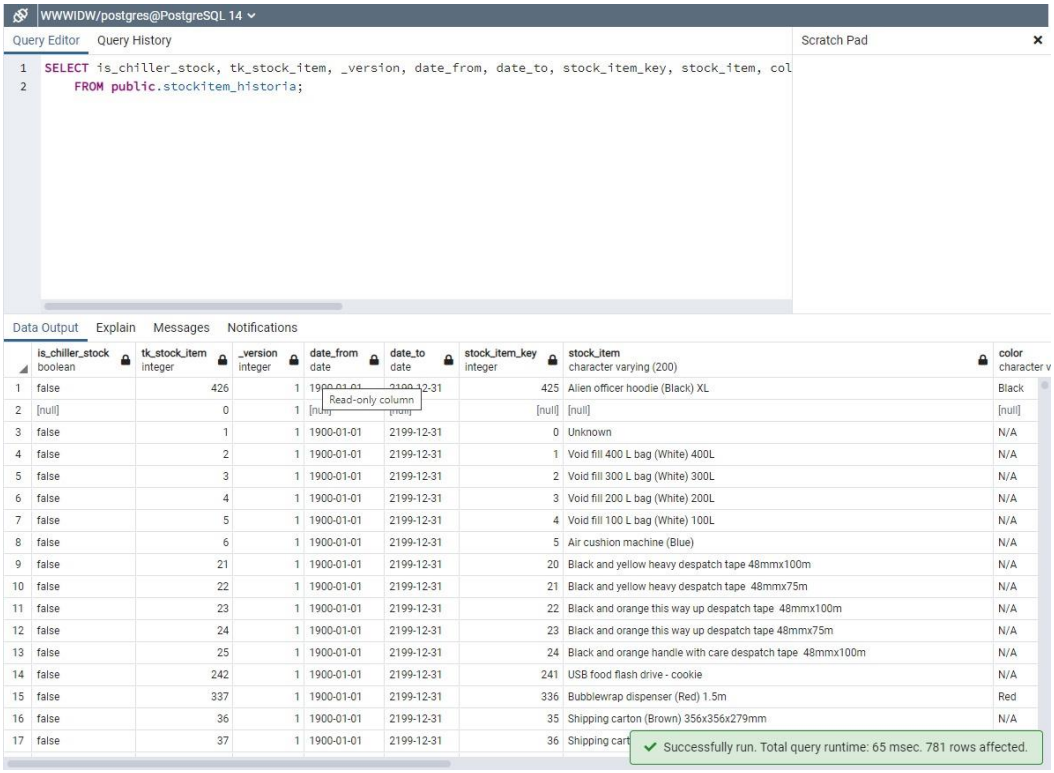
```
SELECT order_key, city_key, customer_key, stock_item_key, order_date_key, picked_date_key, salesperson_key, picker_key, "package", quant
    FROM public.fact_order;
```

Data Output | Explain | Messages | Notifications

	order_key [PK] integer	city_key integer	customer_key integer	stock_item_key integer	order_date_key date	picked_date_key date	salesperson_key integer	picker_key integer	package character varying (50)	quantity integer	unit_p numer
1	2	83	2	631	2015-11-17	2013-07-19	2	133	S	76	
2	23	68	221	215	2014-05-16	2014-05-06	54	156	S	919	
3	24	51	382	399	2016-06-18	2016-02-26	186	51	S	589	
4	25	82	78	336	2014-09-08	2013-01-21	74	188	L	92	
5	26	19	19	15	2016-01-16	2014-04-20	87	191	S	180	
6	27	70	182	366	2016-08-05	2014-01-05	105	201	XL	873	
7	28	39	342	499	2013-12-13	2016-10-21	185	153	S	275	
8	29	75	115	292	2013-06-21	2014-05-16	76	179	S	627	
9	30	13	347	61	2014-12-22	2016-10-31	147	88	M	726	
10	31	50	122	13	2015-08-26	2016-02-27	136	156	S	245	
11	32	46	144	372	2014-06-16	2013-05-12	187	146	S	534	
12	33	34	65	351	2014-12-30	2016-10-10	94	162	S	944	
13	34	79	13	637	2016-06-14	2014-02-27	33	168	XL	120	
14	35	67	348	192	2016-02-18	2015-05-01	122	57	M	802	
15	36	82	235	267	2014-06-26	2015-03-14	103	86	S	337	
16	37	61	289	428	2014-10-15	2013-01-20	108	189	M	590	
17	38	20	255	332	2015-11-03	2014-03-21					

Successfully run. Total query runtime: 106 msec. 1030 rows affected.

Stock Item:



```
SELECT is_chiller_stock, tk_stock_item, _version, date_from, date_to, stock_item_key, stock_item, col
    FROM public.stockitem_historia;
```

	is_chiller_stock boolean	tk_stock_item integer	_version integer	date_from date	date_to date	stock_item_key integer	stock_item character varying (200)	color character v
1	false	426	1	1900-01-01	2199-12-31	425	Alien officer hoodie (Black) XL	Black
2	[null]	0	1	[null]	[null]	[null]	[null]	[null]
3	false	1	1	1900-01-01	2199-12-31	0	Unknown	N/A
4	false	2	1	1900-01-01	2199-12-31	1	Void fill 400 L bag (White) 400L	N/A
5	false	3	1	1900-01-01	2199-12-31	2	Void fill 300 L bag (White) 300L	N/A
6	false	4	1	1900-01-01	2199-12-31	3	Void fill 200 L bag (White) 200L	N/A
7	false	5	1	1900-01-01	2199-12-31	4	Void fill 100 L bag (White) 100L	N/A
8	false	6	1	1900-01-01	2199-12-31	5	Air cushion machine (Blue)	N/A
9	false	21	1	1900-01-01	2199-12-31	20	Black and yellow heavy despatch tape 48mmx100m	N/A
10	false	22	1	1900-01-01	2199-12-31	21	Black and yellow heavy despatch tape 48mmx75m	N/A
11	false	23	1	1900-01-01	2199-12-31	22	Black and orange this way up despatch tape 48mmx100m	N/A
12	false	24	1	1900-01-01	2199-12-31	23	Black and orange this way up despatch tape 48mmx75m	N/A
13	false	25	1	1900-01-01	2199-12-31	24	Black and orange handle with care despatch tape 48mmx100m	N/A
14	false	242	1	1900-01-01	2199-12-31	241	USB food flash drive - cookie	N/A
15	false	337	1	1900-01-01	2199-12-31	336	Bubblewrap dispenser (Red) 1.5m	Red
16	false	36	1	1900-01-01	2199-12-31	35	Shipping carton (Brown) 356x356x279mm	N/A
17	false	37	1	1900-01-01	2199-12-31	36	Shipping cart	

Read-only column

Successfully run. Total query runtime: 65 msec. 781 rows affected.

Resultados en Python (segunda herramienta seleccionada)

Stock Item:

Se escriben los valores en la base de datos del Dataframe generado:

```
# se guardan las historias en un csv
stockitem_historia.to_csv('stockitem_historia.csv', encoding="utf-8", index=False)
```

```
# se insertan los valores de las historias en la base de datos, en este caso en la tabla stockitem_historia
stockitem_historia.to_sql('stockitem_historia',con=cnx,if_exists='append',index=False)
```

Luego leemos los valores de la base de datos para verificar que todo quede manera adecuada:

```
# se lee la tabla en la base de datos par comprobar que todo esté bien
stockitem_historia=pd.read_sql_query('select * from stockitem_historia',cnx)
stockitem_historia = stockitem_historia.sort_values(['stock_item_key', 'date_from'])
stockitem_historia.head(20)
```

nit_price	recommended_retail_price	typical_weight_per_unit	Version	date_from	date_to	recommended_retail_Price	typical_Weight_Per_Unit
50.0	75.0	1.0	1	1900-01-01	2021-11-20	None	None
50.00	75.0	1.0	2	2021-11-20	2199-12-31	74.75	1.000
38.0	56.0	1.0	1	1900-01-01	2021-11-20	None	None
37.50	56.0	1.0	2	2021-11-20	2199-12-31	56.06	.750
25.0	37.0	0.0	1	1900-01-01	2021-11-20	None	None

Al efectuar una query a la base de datos, vemos como se agrega el atributo de Versión y, adicionalmente, date_from y date_to. Esto confirma que la implementación de versionamiento quedó adecuada.

Fact Order:

Se escribe el resultado en la base de datos.

```
# paso 3

df_new_fact = df_empty
# inserción en la tabla de hechos original
df_new_fact.to_sql('fact_order',con=cnx,if_exists='append',index=False)
```

Luego leemos los valores de la base de datos para verificar que todo quede manera adecuada:

```
# lectura de la tabla en la base de datos para comprobar que todo está correcto
df_new_fact=pd.read_sql_query('select * from fact_order',cnx)
df_new_fact.head(20)
```

	order_key	city_key	customer_key	stock_item_key	order_date_key	picked_date_key	salesperson_key	picker_key	package	quantity	unit_pric
0	1	91	179	533	2014-02-05	2013-03-17	135	122	S	805	237.7
1	2	83	2	631	2015-11-17	2013-07-19	2	133	S	76	721.7
2	4	8	218	157	2015-04-04	2014-12-03	84	191	S	878	4671.0
3	5	94	167	235	2015-01-26	2015-01-11	91	197	S	583	1950.6
4	8	54	325	629	2013-10-23	2014-07-01	56	56	M	80	1852.7
5	9	64	263	357	2014-01-29	2015-01-18	184	25	S	777	295.7
6	10	87	236	607	2015-08-10	2014-06-25	71	200	XL	881	2036.8
7	12	56	291	159	2016-04-07	2016-05-31	7	143	L	172	1004.3
8	16	42	58	453	2016-09-08	2016-12-29	35	57	S	776	1737.7
9	21	35	314	195	2013-03-04	2016-04-19	76	87	S	290	3033.0
10	22	73	135	57	2013-04-03	2015-08-25	37	81	S	565	3395.0
11	23	68	221	215	2014-05-16	2014-05-06	54	156	S	919	4158.0
12	24	51	382	399	2016-06-18	2016-02-26	186	51	S	589	4703.9

Customer:

Tipo I:

Escribimos en la base de datos el Dataframe generado. En esta implementación solamente se sobrescriben los registros.

```
# se escribe en la base de datos el conjunto actualizado
df_customer.to_sql('customer',con=cnx,if_exists='append',index=False)
```

12	13	Tailspin Toys (Stonefort- IL)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Razeena Hosseini	90685.0
13	12	Tailspin Toys (Biscay- MN)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Heloisa Fernandes	90054.0
14	15	Tailspin Toys (Batson- TX)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Filips Jaunzems	90631.0
15	14	Tailspin Toys (Long Meadow- MD)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Tereza Valentova	90633.0
16	17	Tailspin Toys (East Fultonham- OH)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Adam Kubat	90416.0
17	16	Tailspin Toys (Coney Island- MO)	Tailspin Toys (Head Office)	Toys Shop	Tailspin Toys	Nitin Matondkar	90467.0
18	19	Tailspin Toys (Lemeta- AK)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Mithun Bhattacharya	90303.0
19	18	Tailspin Toys (Goffstown- NH)	Tailspin Toys (Head Office)	Toys Shop	Tailspin Toys	Isabelle Vodian	90321.0

Tipo II:

Escribimos los valores en la base de datos:

```
# se guardan las historias en un csv
df_customer_historia.to_csv('customer_historia.csv', encoding="utf-8", index=False)

# se insertan los valores de las historias en la base de datos, en este caso en la tabla stockitem_historia
df_customer_historia.to_sql('customer_historia',con=cnx,if_exists='append',index=False)
```

Verificamos que los datos escritos en la base de datos hayan sido correctos:

```
# se lee la tabla en la base de datos par comprobar que todo esté bien
df_customer_historia = pd.read_sql_query('select * from customer_historia',cnx)
df_customer_historia = df_customer_historia.sort_values(['customer_key', 'date_from'])
df_customer_historia.head(20)
```

	tk_stock_item	customer_key	customer	bill_to_customer	category	buying_group	primary_contact	postal_code	Version	date_from	date_to
0	1	1	Tailspin Toys (Head Office)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Waldemar Fisar	90410	1	1900-01-01	2021-11-20
1	2	1	Tailspin Toys (Head Office)	Tailspin Toys (Head Office)	Toys Shop	Tailspin Toys	Waldemar Fisar	90410	2	2021-11-20	2199-12-31
2	3	2	Tailspin Toys (Sylvanite-MT)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Lorena Cindric	90216	1	1900-01-01	2199-12-31
3	4	3	Tailspin Toys (Peeples Valley- AZ)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Bhaargav Rambhatla	90205	1	1900-01-01	2199-12-31
4	5	4	Tailspin Toys (Medicine Lodge- KS)	Tailspin Toys (Head Office)	Novelty Shop	Tailspin Toys	Daniel Roman	90152	1	1900-01-01	2199-12-31

En el tipo 2 se encuentra los atributos de versión y las fechas de actualización.

Tipo III:

Escribimos los valores en la base de datos:

```
# se guardan las historias en un csv
df_customer_historia.to_csv('customer_historia.csv', encoding="utf-8", index=False)

# se insertan los valores de las historias en la base de datos, en este caso en la tabla stockitem_historia
df_customer_historia.to_sql('customer_historia',con=cnx,if_exists='append',index=False)
```

Verificamos que los datos escritos en la base de datos hayan sido correctos:

```
# se lee la tabla en la base de datos par comprobar que todo esté bien
df_customer_historia = pd.read_sql_query('select * from customer_historia',cnx)
df_customer_historia = df_customer_historia.sort_values(['customer_key', 'date_from'])
df_customer_historia.head(20)
```

	customer_key	customer	bill_to_customer	buying_group	primary_contact	postal_code	previous_category	new_category	date_updated
0	1	Tailspin Toys (Head Office)	Tailspin Toys (Head Office)	Tailspin Toys	Waldemar Fisar	90410	Novelty Shop	Toys Shop	2021-11-20
1	2	Tailspin Toys (Sylvanite- MT)	Tailspin Toys (Head Office)	Tailspin Toys	Lorena Cindric	90216	NAN	Novelty Shop	2021-11-20
2	3	Tailspin Toys (Peeples Valley- AZ)	Tailspin Toys (Head Office)	Tailspin Toys	Bhaargav Rambhatla	90205	NAN	Novelty Shop	2021-11-20
3	4	Tailspin Toys (Medicine Lodge- KS)	Tailspin Toys (Head Office)	Tailspin Toys	Daniel Roman	90152	NAN	Novelty Shop	2021-11-20
4	5	Tailspin Toys (Gasport- NY)	Tailspin Toys (Head Office)	Tailspin Toys	Johanna Huiting	90261	NAN	Novelty Shop	2021-11-20
5	6	Tailspin Toys (Jessie- ND)	Tailspin Toys (Head Office)	Tailspin Toys	Biswajeet Thakur	90298	NAN	Novelty Shop	2021-11-20

Comparación de herramientas de ETL

Spoon	Python
Pros: -Es una herramienta bastante potente porque permite combinar las sentencias SQL desde la herramienta para creación de las tablas y su base de datos. -Permite modelar tareas complejas. -Es bastante intuitivo para efectuar transformaciones sobre los datos. **Contras:** -Permite integrar con una limitada cantidad de bases de datos, aun así, cumple con las principales. -Tiene una interfaz que puede ser poco intuitiva para el usuario en una primera instancia. -Presenta muchos errores por lecturas equivocadas sobre los tipos de datos en un .CSV. Por lo que es necesario realizar correcciones manualmente que consumen bastante tiempo. -Solo acepta lectura de datos en formato UTF-8. -La limpieza de datos debe ser efectuada desde un Software externo, puesto que su implementación en Spoon es bastante compleja.	**Pros:** -Presenta librerías para conexión SQL con prácticamente cualquier base de datos. -Pandas y Numpy son particularmente útiles para el preprocesamiento y transformación del conjunto de datos. -Los Dataframes permiten hacer un manejo más elaborado sobre las dimensiones, puesto que no se necesitan realizar Querys o sentencias bastante complicadas. -Acepta lecturas de datos en prácticamente cualquier forma y se pueden convertir los mismos. **Contras:** -Al ser un lenguaje de alto nivel tiene se requiere de un mínimo de conocimiento previo para su uso. -La interfaz es exclusivamente a través de líneas de código, por lo que no presenta un CLI intuitivo al usuario. -Se debe especificar muy bien el tipo para cada atributo en una tabla de hechos o dimensión, siendo este un proceso tedioso y demorado. -No tiene claras las restricciones SQL en los modelos, e implementarlas puede llegar a ser un proceso de muchas líneas de código.

Preguntas

- **¿Cuál es el objetivo de la columna tk_stock_item?**
  Al crear la historia de stock_item, las referencias hacia Stock_Item pueden repetirse, por lo tanto, es necesario implementar una forma de referenciar para objetos únicos; y, esta forma de referenciar es la llave subrogada tk_stock_item para referenciar a cada una de las historias.

- **¿Qué significa cada una de estas opciones?**
  - Insert: implementar un manejo de historia Tipo II. Si se detecta una diferencia para uno o más mapeos que tienen la opción Insert, se añade una fila a la tabla de dimensión.
  - Update: se actualiza la fila. Se puede utilizar para implementar el manejo de historia Tipo I.

  - Punch through: también realiza una actualización. En lugar de solo actualizar la fila, se actualizan todas las versiones de la fila, aplicando un manejo de historia Tipo II.
  - Date of last insert or update (without sream field as source): permite mantener un campo de fecha automático que registra la fecha del insert o el update.
  - Date of last insert (without stream field as source): permite mantener un campo de fecha automático que registra la fecha del último insert.
  - Date of last update (without stream field as source): permite mantener un campo de fecha automático que registra la fecha del último update.
  - Last versión (without stream field as source): permite mantener una 'flag' automática que indica si la fila es la última versión.

- **¿Cómo se puede saber que el proceso ETL manejó los cambios entre los dos archivos?**
  En primer lugar, se puede observar que la llave primaria del Stock_Item se está repitiendo, esto indica que hubo un versionamiento sobre este registro; adicionalmente, este versionamiento se puede ver reflejado en el número de registro y en las fechas de Date_From y Date_To.

- **¿Por qué se debe hacer el cambio para que la columna de la llave subrogada sea de tipo Unique?**
  Porque en versionamiento no puede existir una repetición en el número de registro. Si se presenta repetición, el negocio no sabría con claridad qué modificaciones fueron presentadas en el conjunto de datos. También, búsquedas específicas relacionadas con versionamiento, se pueden ver afectadas si la llave subrogada no es única.

- **¿Qué pasa si uno de los datos reportados en la tabla de hechos no existe en alguna de las dimensiones?**
  Como la tabla de hechos se encuentra compuesta de llaves foráneas sobre las dimensiones, al no existir un dato reportado, no se presenta una referencia entre la tabla de hechos y la dimensión, por lo que es imposible generar el registro. Esta situación presenta bota un error al momento de cargar los datos en la tabla de hechos.

- **¿Qué sugiere para evitar esa situación?**
  Al momento de insertar valores en la tabla de hechos, se puede verificar de manera previa si las relaciones que se pretenden agregar con respecto a las dimensiones existen (ya sea por medio de una Query u otro tipo de llamados al conjunto de datos). Esto evita que se ingresen valores no permitidos a la tabla de hechos.

Enlace del repositorio de anexos

A continuación, se presenta el repositorio que muestra todos los resultados obtenidos, las tablas utilizadas y los proyectos implementados:

https://github.com/SergioZona/Lab5_ee_ortiz_mc_parrad_sj_zona