1. **Can you describe the file format?**

The data is pipe separated. There are total 8 files from TPC-H dataset.

1. **Super Bonus: generate your own data through the instructions on the encoded file bonus\_etl\_data\_gen.txt**

I have created my own dataset after decoding the instruction at “bonus\_etl\_data\_gen.txt” and I have attached my encoded step details in the file “how\_did\_i\_generate\_my\_own\_dataset.txt”

1. **Code you scripts to load the data into a database**

Although I used the challenge steps to get a SQLITE database with all the tables in a single file, to load the data from the files, I would have used the steps below.

sqlite3 test.db

-- create the tables using supplied DDLs

.separator '|'

.import "C:\\interview-test-data-engineer-master\\data\\customer.tbl" customer

.import "C:\\interview-test-data-engineer-master\\data\\NATION.tbl" NATION

.import "C:\\interview-test-data-engineer-master\\data\\REGION.tbl" region

.import "C:\\interview-test-data-engineer-master\\data\\PART.tbl" PART

.import "C:\\interview-test-data-engineer-master\\data\\SUPPLIER.tbl" SUPPLIER

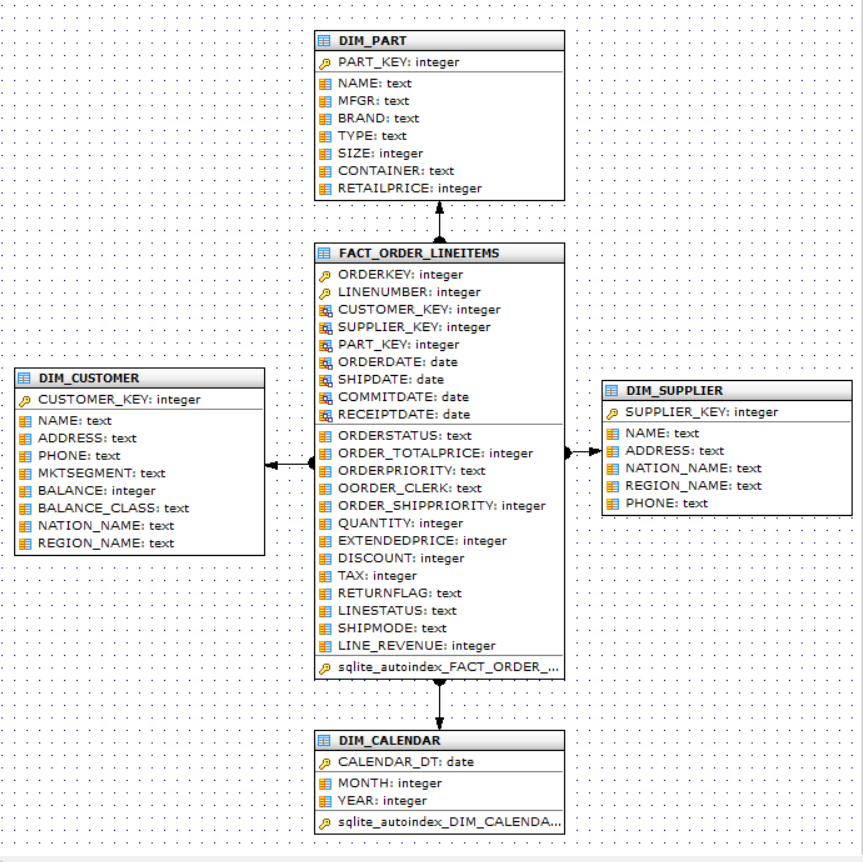
.import "C:\\interview-test-data-engineer-master\\data\\PARTSUPP.tbl" PARTSUPP

.import "C:\\interview-test-data-engineer-master\\data\\ORDERS.tbl" ORDERS

.import "C:\\interview-test-data-engineer-master\\data\\LINEITEM.tbl" LINEITEM

1. **Design a star schema model which the data should flow**

The DDL to create the star schema is available in the file “ddl - star.sql”



1. **Build your process to load the data into the star schema**

The scripts to load the star schema is available in the file “**prospa\_dw\_etl.py**”. Following assumptions were made during the ETL design for simplification:

1. Dimensions were considered as TYPE-1 dimension. i.e. no historical tracking of slowly changing dimensional attributes. Its assumed that business does not require to report dimensions at a particular point in time. So, the Natural key was used as primary key in the DW dimension table. In ideal situation, dimension tables will have their own surrogate keys.
2. No CDC technique was applied for ETL. The ETL scans through the whole data from the source tables and detects whether to insert or update the records in DW based on the keys. However, ETLs were designed as idempotent i.e. the ETLs can be run multiple times on the same or changed dataset again and again without causing data integration issues.
3. To keep the ETL idempotent, methodology was followed to load dimension and fact tables using natural keys. Check the existence of the key in the DW table. If it exists, update all the columns. If the key does not exist, insert the record into the DW tables.
4. **Bonus point:**
   1. **add a fields to classify the customer account balance in 3 groups**

Added a column named BALANCE\_CLASS in DIM\_CUSTOMER table with the following 3 groups: ‘Negative’, ‘0-5000’, and ‘>5000’

* 1. **add revenue per line item**

Added a column named LINE\_REVENUE in the table FACT\_ORDER\_LINEITEMS with the following formula (QUANTITY\*EXTENDEDPRICE \* (1- DISCOUNT))

* 1. **convert the dates to be distributed over the last 2 years**

1. **How to schedule this process to run multiple times per day?**

Since the etl is a simple python job in a single python file, the simplest solution will be to use OS schedulers (e.g. CRON, task scheduler in Windows) to run at particular intervals.

1. **Bonus: What to do if the data arrives in random order and times via streaming?**

This simple ETL implementation does not handle late arriving or other complex situations. Ideally, there will be a persistent staging layer where data will be keep appending as it arrives from the source. Then the further ETL to load the star schema will consider the latest record per key from the persistent stage table to construct the dimension and fact tables.

There are several strategies to handle late arriving dimension and fact e.g. withholding processing of the records, assigning the surrogate key for Unknown dimensional value e.g. 0 to denote that the dimensional values were not available. Those records can be updated again once the records are available.

1. **How to deploy this code? Bonus: Can you make it to run on a container like process (Docker)?**

The instruction to deploy the code was mentioned at the beginning of this document.

# Data Reporting

1. **What are the top 5 nations in terms of revenue?**

Ans:

CANADA|105337574.5622

EGYPT|102254394.9985

IRAN|100283451.6143

BRAZIL|94333196.6970001

ALGERIA|93680675.2906

select DIM\_CUSTOMER.NATION\_NAME, sum(FACT\_ORDER\_LINEITEMS.LINE\_REVENUE)

from FACT\_ORDER\_LINEITEMS

inner join DIM\_CUSTOMER on FACT\_ORDER\_LINEITEMS.CUSTOMER\_KEY = DIM\_CUSTOMER.CUSTOMER\_KEY

group by DIM\_CUSTOMER.NATION\_NAME

order by 2 desc

limit 5;

1. **From the top 5 nations, what is the most common shipping mode?**

**Ans: FOB with 2118 orders**

select SHIPMODE, count(\*)

from FACT\_ORDER\_LINEITEMS

join DIM\_CUSTOMER on FACT\_ORDER\_LINEITEMS.CUSTOMER\_KEY = DIM\_CUSTOMER.CUSTOMER\_KEY

where DIM\_CUSTOMER.NATION\_NAME in

(

select NATION\_NAME from

(

select DIM\_CUSTOMER.NATION\_NAME, sum(FACT\_ORDER\_LINEITEMS.LINE\_REVENUE)

from FACT\_ORDER\_LINEITEMS

inner join DIM\_CUSTOMER on FACT\_ORDER\_LINEITEMS.CUSTOMER\_KEY = DIM\_CUSTOMER.CUSTOMER\_KEY

group by DIM\_CUSTOMER.NATION\_NAME

order by 2 desc

limit 5

)

)

group by SHIPMODE

order by 2 desc

;

1. **What are the top selling months?**

Ans: Top 5 selling months based on revenue below:

05|186910950.3546

03|183780987.1041

01|180816143.6486

07|178259733.6994

04|177213598.997999

select strftime('%m',FACT\_ORDER\_LINEITEMS.ORDERDATE), sum(FACT\_ORDER\_LINEITEMS.LINE\_REVENUE)

from FACT\_ORDER\_LINEITEMS

group by strftime('%m',FACT\_ORDER\_LINEITEMS.ORDERDATE)

order by 2 desc

limit 5;

1. **Who are the top customer in terms of revenue and/or quantity?**

**--Top 5 Customer based on revenue**

Customer#000001489|5203674.0537|3868

Customer#000000214|4503703.9036|3369

Customer#000000073|4466381.0513|3384

Customer#000001246|4465335.6222|3226

Customer#000001396|4455381.8182|3408

select DIM\_CUSTOMER.NAME, sum(FACT\_ORDER\_LINEITEMS.LINE\_REVENUE), sum(QUANTITY)

from FACT\_ORDER\_LINEITEMS

inner join DIM\_CUSTOMER on FACT\_ORDER\_LINEITEMS.CUSTOMER\_KEY = DIM\_CUSTOMER.CUSTOMER\_KEY

group by DIM\_CUSTOMER.NAME

order by 2 desc

limit 5;

-- Top 5 customers based on Quantity

Customer#000001489|5203674.0537|3868

Customer#000001396|4455381.8182|3408

Customer#000000073|4466381.0513|3384

Customer#000000214|4503703.9036|3369

Customer#000000898|4305984.9017|3309

select DIM\_CUSTOMER.NAME, sum(FACT\_ORDER\_LINEITEMS.LINE\_REVENUE), sum(QUANTITY)

from FACT\_ORDER\_LINEITEMS

inner join DIM\_CUSTOMER on FACT\_ORDER\_LINEITEMS.CUSTOMER\_KEY = DIM\_CUSTOMER.CUSTOMER\_KEY

group by DIM\_CUSTOMER.NAME

order by 3 desc

limit 5;

1. **Compare the sales revenue of on current period against previous period?**

select current.CURRENT\_PERIODD, current.total\_revenue revenue\_current\_period, last\_year.total\_revenue revenue\_same\_period\_last\_year

from

(

select strftime('%Y%m',FACT\_ORDER\_LINEITEMS.ORDERDATE) CURRENT\_PERIODD, strftime('%Y%m',date(FACT\_ORDER\_LINEITEMS.ORDERDATE,'-1 years')) same\_PERIODD\_last\_year, sum(FACT\_ORDER\_LINEITEMS.LINE\_REVENUE) TOTAl\_REVENUE

from FACT\_ORDER\_LINEITEMS

group by strftime('%Y%m',FACT\_ORDER\_LINEITEMS.ORDERDATE),strftime('%Y%m',date(FACT\_ORDER\_LINEITEMS.ORDERDATE,'-1 years'))

) current

left outer join

(

select strftime('%Y%m',FACT\_ORDER\_LINEITEMS.ORDERDATE) same\_PERIODD\_last\_year, sum(FACT\_ORDER\_LINEITEMS.LINE\_REVENUE) TOTAl\_REVENUE

from FACT\_ORDER\_LINEITEMS

group by strftime('%Y%m',FACT\_ORDER\_LINEITEMS.ORDERDATE)

) last\_year

on current.same\_PERIODD\_last\_year = last\_year.same\_PERIODD\_last\_year

order by 1 desc

;

PERIOD|PERIOD\_REVENUE|SAME\_PERIOD\_LAST\_YEAR\_REVENUE

199808|1515799.5822|25469404.176

199807|26283121.969|25000779.4059

199806|23423472.1215|25064490.4757

199805|28145721.0172|26202916.1322

199804|24837949.6163|26229550.586

199803|27707128.3949|26951768.0446999

199802|24535858.6301|26202133.8903

199801|23556175.4298|27189203.2127

199712|24691135.1631|28317340.1076

199711|26824996.9882|27353044.7232

199710|24019551.2011|23461944.6939

199709|24146710.2135|27042962.0044

199708|25469404.176|28974470.7184

199707|25000779.4059|25544543.6775

199706|25064490.4757|27377693.1476

199705|26202916.1322|22333464.0534

199704|26229550.586|25391680.8265

199703|26951768.0446999|27070929.1023

199702|26202133.8903|24708562.3861

199701|27189203.2127|24351722.3396

199612|28317340.1076|28896188.4313

199611|27353044.7232|27428356.0733

199610|23461944.6939|27167545.7641

199609|27042962.0044|24422897.5107

199608|28974470.7184|24359286.7382

199607|25544543.6775|26045908.5861

199606|27377693.1476|23876409.1967

199605|22333464.0534|27820082.022

199604|25391680.8265|23619272.5549

199603|27070929.1023|23352432.7438

199602|22626564.1951|23987366.5956

199602|2081998.191|23352432.7438

199601|24351722.3396|22934747.8854

199512|28896188.4313|27568721.0927

199511|27428356.0733|22829061.7679

199510|27167545.7641|25175323.7622

199509|24422897.5107|28325144.2007

199508|24359286.7382|27816789.746

# Data profilling

1. **What tools or techniques you would use to profile the data?**

pandas-profiling is a nice utility to get useful information on all data elements e.g. distribution, min, max, outliers etc.

1. **What results of the data profiling can impact on your analysis and design?**

Data profiling can help understand the potential data issues and help developers to handle the scenarios.

# Architecture

**If this pipeline is to be build for a real live environment. What would be your recommendations in terms of tools and process?**

For a real live environment, I would recommend the following systems in place for optimal processing:

1. A CDC log capturing tool which can efficiently generate CDC data as data gets changed
2. CDC data to be appended in a persistent staging area where individual records will be immutable to provide auditability.