Hevo Data Interview Exercise Steps Executed :

Installation of docker & postgresql instance :

1. Install the docker application from the web onto the system.
2. Once the application is installed pull the postgre image from the container using the below command:

* *docker pull postgres*

1. On the next step create a data volume which will hold all the necessary files associated with the postgre container. Use the below command :

* *docker volume create postgres-data*

1. Once the data volume is created run the container to create a postgresql instance by using the below command :

* *docker run --name postgres-container -e POSTGRES\_PASSWORD=\*\*\*\*\*\* -p 5432:5432 -v postgres-data:/var/lib/postgresql/data -d postgres*

1. Use the psql container command to create any database or table. Use the below command :

* *docker exec -it postgres-container psql -U postgres*

1. For this exercise I used dbeaver to import the files into postgre as it was a much simpler effort in comparison to pgadmin4. Imported all the files and used create utility to create RAW tables in postgre.
2. Next step whitelisted all the hevo data ip’s to build the pipeline in hevo. This can be achieved by making changes to the below files using the command

* *docker exec -it postgres-container bash*
* *hba\_conf – use the whitelisted ip address provided in Hevo or 0.0.0.0/0 or all and keep the method as md5*

*# IPv4 local connections:*

*host all all all md5*

* *postgresql.conf – change the listen address to \**

1. Restart the container using command docker container restart

Setting up Snowflake & Hevo Partner Account :

1. Login to Snowflake and create a 30 day free trial account. Setup the snowflake account
2. From the apps section select hevo and configure the hevo account.

Setting up the Hevo Data Pipeline :

1. Go to the homebrew page and use the below command :

* */bin/bash -c "$(curl -fsSL* [*https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh*](https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)*)"*

1. Once brew is installed run the below commands :

* *brew install ngrok/ngrok/ngrok – install ngrok*
* *ngrok config add-authtoken 2ok0QDN0DfB5X6PuvSncDyvjMGh\_6qXCNnEC5HEQfaeJQJGMM – create the authorization token*
* *ngrok tcp 5432 – use it to host the 5432 postgre port*

1. Once the localhost is online, capture the forwarding details :

* *tcp://0.tcp.in.ngrok.io:18262*

1. Move to the Hevo Data UI, start building the pipeline use the source connector as postgre. Once done give the pipeline name, hostname becomes *tcp://0.tcp.in.ngrok.io and port becomes 18262*
2. Test the connection and move ahead choosing logical replication method.
3. Next will be data ingestion step which hevo will do it automatically by selecting the required object i.e the raw tables created from the source files(customer, order & payments). Run the pipeline.
4. Once data ingestion is complete in the target step the Snowflake connector automatically pops up since it’s a partner connect, select that and complete the process. Provide proper schema names and table names
5. Check if the tables got populated in the correct schema, database and tables in Snowflake. The data will be already be populated once the pipeline has run successful.

**Setting up dbt with Snowflake :**

1. Login to dbt cloud and create an account.
2. Once the account gets created, in the next step setup the snowflake connection.
3. Select the connection type as Snowflake and the provide the below details :

* *Account : yp12027.central-india.azure*
* *Database : PC\_HEVODATA\_DB*
* *Warehouse : COMPUTE\_WH*

1. Provide the username and password. After that test and setup the connection.
2. Once the connection is setup go to the develop and select Cloud IDE.
3. Select the Snowflake connection and select the github account on which the codes will get uploaded.
4. On the file explorer tab, create the following two files sources.yml and customer.sql. The sources.yml will have the source details such as database name, schema names, column for which any native tests needs to be built. In this case the native test cases have been built in the sources.yml to prevent creating of new model files.

* Sources.yml :

version: 2

sources:

name: hevo\_project

database: PC\_HEVODATA\_DB

schema: public

tables:

- name: SNOW\_RAW\_PS\_RAW\_CUSTOMERS

columns:

- name: id

tests:

- not\_null

- unique

- name: SNOW\_RAW\_PS\_RAW\_ORDERS

columns:

- name: id

tests:

- not\_null

- unique

- name: user\_id

tests:

- not null

- name: SNOW\_RAW\_PS\_RAW\_PAYMENTS

columns:

- name: id

tests:

- not\_null

- unique

1. In the customer.sql write the equivalent snowflake query to build the materialized table customers.

* Snowflake Query :

*WITH CTE1 AS*

*(*

*SELECT C.ID AS CUSTOMER\_ID,*

*C.FIRST\_NAME AS FIRST\_NAME,*

*C.LAST\_NAME AS LAST\_NAME,*

*FIRST\_VALUE(O.ORDER\_DATE) OVER (PARTITION BY C.ID ORDER BY O.ORDER\_DATE ASC) AS FIRST\_ORDER,*

*LAST\_VALUE(O.ORDER\_DATE) OVER (PARTITION BY C.ID ORDER BY O.ORDER\_DATE ASC) AS LAST\_ORDER,*

*FROM*

*SNOW\_RAW\_PS\_RAW\_CUSTOMERS AS C*

*INNER JOIN SNOW\_RAW\_PS\_RAW\_ORDERS O*

*ON*

*C.ID=O.USER\_ID*

*INNER JOIN SNOW\_RAW\_PS\_RAW\_PAYMENTS P*

*ON O.ID=P.ORDER\_ID),*

*CTE2 AS*

*(*

*SELECT O.USER\_ID AS USER\_ID,*

*COUNT(O.ID) AS NUMBER\_OF\_ORDERS,*

*SUM(P.AMOUNT) AS CUSTOMER\_LIFETIME\_VALUE*

*FROM*

*SNOW\_RAW\_PS\_RAW\_ORDERS O*

*INNER JOIN SNOW\_RAW\_PS\_RAW\_PAYMENTS P*

*ON O.ID=P.ORDER\_ID*

*GROUP BY 1)*

*SELECT \* FROM CTE1 JOIN CTE2*

*ON CTE1.CUSTOMER\_ID=CTE2.USER\_ID;*

* Customers.sql file

*{{*

*config(*

*materialized="table",*

*alias="customer",*

*schema="public"*

*)*

*}}*

*WITH CTE1 AS*

*(*

*SELECT C.ID AS CUSTOMER\_ID,*

*C.FIRST\_NAME AS FIRST\_NAME,*

*C.LAST\_NAME AS LAST\_NAME,*

*FIRST\_VALUE(O.ORDER\_DATE) OVER (PARTITION BY C.ID ORDER BY O.ORDER\_DATE ASC) AS FIRST\_ORDER,*

*LAST\_VALUE(O.ORDER\_DATE) OVER (PARTITION BY C.ID ORDER BY O.ORDER\_DATE ASC) AS LAST\_ORDER,*

*FROM*

*{{ source('hevo\_project','SNOW\_RAW\_PS\_RAW\_CUSTOMERS') }} C*

*INNER JOIN {{ source('hevo\_project','SNOW\_RAW\_PS\_RAW\_ORDERS') }} O*

*ON*

*C.ID=O.USER\_ID*

*INNER JOIN {{ source('hevo\_project','SNOW\_RAW\_PS\_RAW\_PAYMENTS') }} P*

*ON O.ID=P.ORDER\_ID),*

*CTE2 AS*

*(*

*SELECT O.USER\_ID AS USER\_ID,*

*COUNT(O.ID) AS NUMBER\_OF\_ORDERS,*

*SUM(P.AMOUNT) AS CUSTOMER\_LIFETIME\_VALUE*

*FROM*

*{{ source('hevo\_project','SNOW\_RAW\_PS\_RAW\_ORDERS') }} O*

*INNER JOIN {{ source('hevo\_project','SNOW\_RAW\_PS\_RAW\_PAYMENTS') }} P*

*ON O.ID=P.ORDER\_ID*

*GROUP BY 1)*

*SELECT CTE1.CUSTOMER\_ID,CTE1.FIRST\_NAME,CTE1.LAST\_NAME,CTE1.FIRST\_ORDER,CTE1.LAST\_ORDERCTE2.NUMBER\_OF\_ORDERS,CTE2.CUSTOMER\_LIFETIME\_VALUE*

*FROM CTE1 JOIN CTE2 ON CTE1.CUSTOMER\_ID=CTE2.USER\_ID*

1. Run the command dbt build --select customer and let the model execute.
2. Check for any issues if there is any failure. In case there is no failure check in snowflake if the materialized table is created. Table should get created and data should get reflected.
3. Push the changes to git and complete the pipeline.