Awesome Inc: Universal Sales & Returns

Vaidehi Chaudhari (vdc2009)
Amanpreet Singh Saimbhi (as15798)

Course: ECE-GY-9941 Advance Project III

Project Report

Date of Submission: 5/5/2023

TABLE OF CONTENT

ABSTRACT 3

BUSINESS CASE 4

PROJECT MILESTONE 5

Purpose 5
System Requirements 5
Roadmap 6

ASSUMPTIONS 7

OLTP DESIGNS 8

OLTP Logical model 8
OLTP Relational Model 8

DATA WAREHOUSING 9

De-normalization 9
DW Logical Model 9
DW Relational Model 9

ETL SUMMARY 10

SQL & DATA ANALYTICS FOR DATA WAREHOUSE 11

ETL Example with CSV file 11

RECORD COUNT OF OLTP TABLE 16

DATA MANAGEMENT 19

External Table 19
History table 19
Table partitioning 20

INCREMENTAL ETL 21

DATA VISUALIZATION 23

PROJECT INSIGHTS 27

APPENDIX 28

OLTP Data Definition Language Code 28
OLTP Data Manipulation Language Code 36
OLTP Data dictionary 39
OLAP Data Definition Language Code 41
Procedure code for Incremental ETL 46
History table trigger 48
Partitioning of table 48
ETL code 49

ABSTRACT

The goal of this project was to analyze and visualize data using Tableau for effective data-driven decision making. The project involved various tasks, including data extraction, transformation, and loading (ETL), data modeling, and interactive dashboard creation. The data was sourced from multiple data sources and loaded into Snowflake, a cloud-based data warehouse. ETL processes were implemented to clean, transform, and load the data into a suitable format for analysis. Data modeling was done in Snowflake to establish relationships between the data tables and create a foundation for analysis. Finally, interactive dashboards were created using Tableau to visualize the data and provide actionable insights. The dashboards included various charts, tables, and filters, allowing users to explore the data and gain insights into key performance indicators (KPIs). The project successfully achieved the objective of creating an interactive dashboard for data analysis and visualization, which can be used for decision making in the organization.

Keywords: ETL process, Tableau, Snowflake, SQL, data analysis.

BUSINESS CASE

The Universal Sales and Return Data Warehouse and Analytics project aims to centralize sales and return data from multiple sources into a single repository through an ETL process. Currently, Awesome Inc has multiple data sources that contain sales and return data, making it difficult to access and analyze the data effectively. The data is often inconsistent, of low quality, and not easily accessible, leading to inaccurate decision-making and missed business opportunities. This situation creates inefficiencies and can negatively impact the company's bottom line. The proposed solution is to centralize the sales and return data into a single data warehouse using ETL tools such as Oracle Data Modeler. The data will be transformed to ensure consistency, accuracy, and quality before being loaded into the MySQL database.

The transformed data will be analyzed using Tableau for interactive visualizations and dashboards, providing support for data-driven decision-making. The project aims to deliver a comprehensive data warehousing and analytics solution that will enable Awesome Inc to make informed decisions based on accurate and up-to-date data. In addition, the project involves the use of Snowflake, a cloud-based data warehousing platform, for storing and managing large volumes of data. The ETL process involves the extraction of data from different sources, including flat files, databases, and APIs, followed by data cleaning, transformation, and loading into the Snowflake data warehouse. Incremental loading is used to update the data warehouse with new data on a regular basis, ensuring that the data is always current.

The transformed data will be used to create a range of interactive dashboards and visualizations using Tableau, allowing Awesome Inc to gain insights into sales and return trends, customer behavior, and product performance. The project will also enable the creation of ad-hoc reports, enabling quick and easy access to data for business users across the organization.

In short, the Universal Sales and Return Data Warehouse and Analytics project aims to deliver a comprehensive solution that will provide Awesome Inc with a clear view of their sales and return data, enabling informed decision-making and improved business outcomes.

PROJECT MILESTONE

The purpose of this project is to develop a data warehousing solution to extract, transform, and load data into a centralized repository. The goal is to make the data accessible, reliable, and usable for data analytics and reporting. After the ETL process is completed, the project will focus on using Tableau to perform data analytics and gain valuable insights into the data. The outcome of the project is expected to be an enhanced decision-making process by providing a comprehensive understanding of the data.

System Requirements:

✚ Software Requirements:

- Database language: MySQL, Snowflake SQL

Software/Technologies used: Snowflake 3.20.4 , Oracle Data modeler 21.4.2, MySQL Workbench 8.0 CE, AWS S3 bucket, Visual Studio Code, Tableau Desktop 2023.1

✚ Hardware Requirements:

- Windows 10 8GB RAM quad-core processor, Intel i5 processor
- MAC OS X 10.9

Roadmap:

- **Project Proposal:** (23th Jan - 4th Feb)

  The project proposal was submitted along with the purpose, objective, system design.

- **OLTP Design:** (5th Feb - 18thth Feb)

  Logical Database model, Relational Database model, DDL Generation was submitted.

- **OLTP Database Implementation:** (19th Feb - 3rd Mar)

  The database was created in MySQL workbench and tables were populated after loading the staging table from CSV files. Also, history table was created to store the deleted entries.

- **OLAP Design:** (4th Mar- 18thth March)

  Counts of each records were submitted. De-normalized design was created in star schema. Logical model and Relational model of OLAP was submitted.

- **Data Warehouse design and implementation:** (19th Mar - 15th April)

  De- normalization from the OLTP design. Use of AWS bucket for S3 bucket to load CSV

  Use of Snowflake software which is a cloud computing-based data warehousing software.Made use of Snowflake for cloud-based storage and data analytics. Created history table for address.

- **ETL code using Snowflake and AWS:** (16th Apr – 25th Apr)

  The database was created in MySQL workbench and tables were populated after loading the staging table from CSV files. Also, history table was created to store the deleted entries. Created External table in Snowflake. Incremental ETL was also implemented to make sure that the new data is also updated in the data-warehouse.

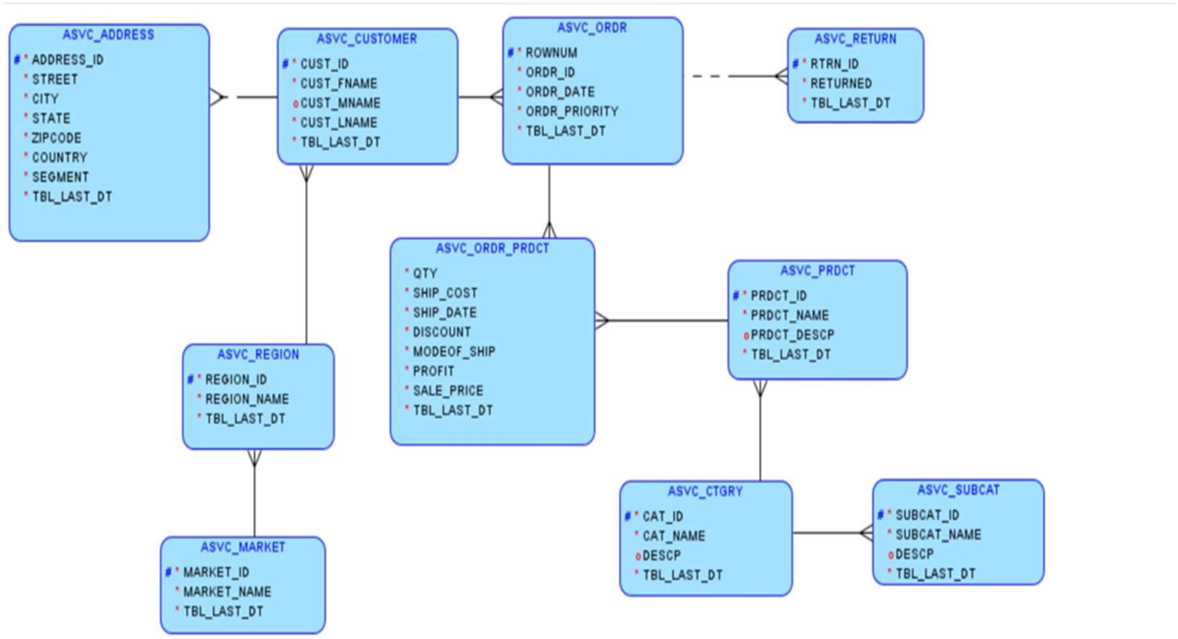
- **Reports and Analytics from DW database:** (26th Apr - 2d Mar)

  Querying of the data and generation of reports and data analytics and data visualization is done by creating dashboards on Tableau Desktop.

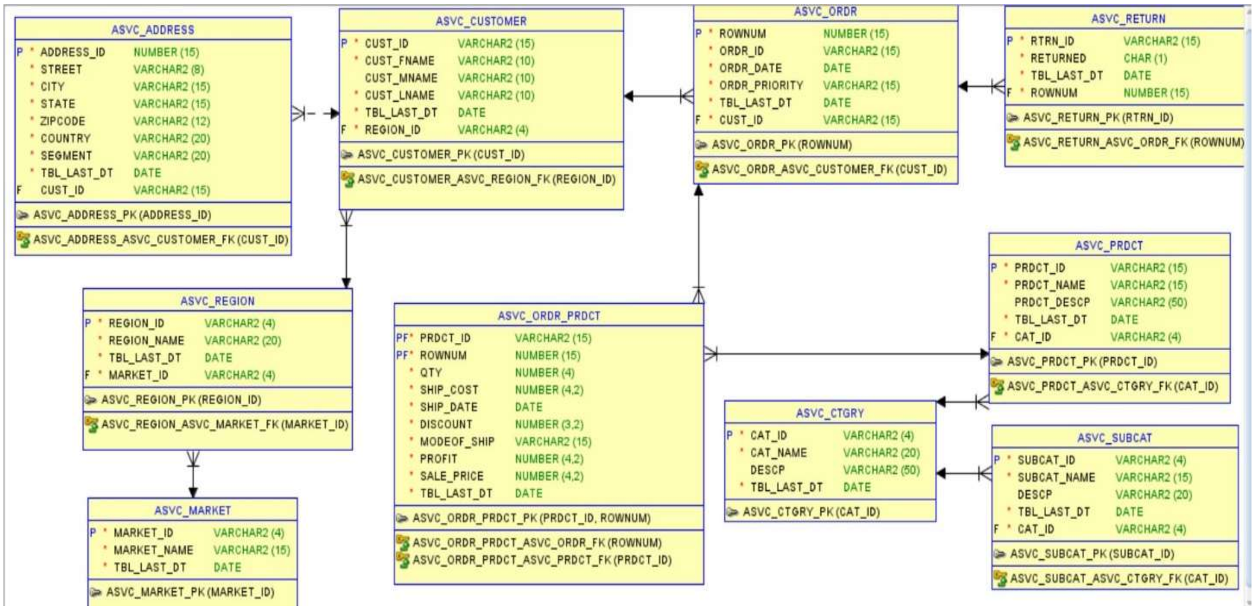
ASSUMPTIONS

1. A customer can have multiple address, but an address belongs to only one customer.
2. A region can have multiple customers, but a customer belongs to only one region.
3. A market can have many regions associated with it but a region will be associated to only one market.
4. Each order is associated with one and only one customer.
5. A customer can place multiple orders.
6. Each order can have multiple products.
7. Each product can be part of multiple orders.
8. Each product belongs to one and only one category.
9. Each category can have multiple products.
10. Each category can have multiple sub-categories.
11. Each sub-category belongs to one and only one category.
12. The region and market details of a customer are based on the shipping address of their orders.
13. The sales price, shipping cost, and profit details are associated with each individual product in an order.

OLTP LOGICAL MODEL



OLTP RELATIONAL MODEL

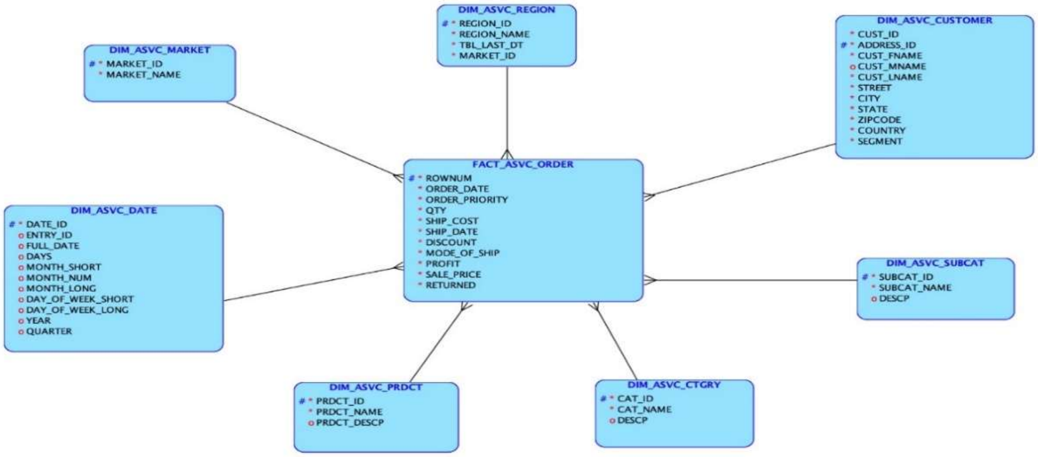

ASVC_ADDRESS
P * ADDRESS_ID NUMBER (15)
* STREET VARCHAR2 (8)
* CITY VARCHAR2 (15)
* STATE VARCHAR2 (15)
* ZIPCODE VARCHAR2 (12)
* COUNTRY VARCHAR2 (20)
* SEGMENT VARCHAR2 (20)
* TBL_LAST_DT DATE
F CUST_ID VARCHAR2 (15)
ASVC_ADDRESS_PK (ADDRESS_ID)
ASVC_ADDRESS_ASVC_CUSTOMER_FK (CUST_ID)
ASVC_CUSTOMER
P * CUST_ID VARCHAR2 (15)
* CUST_FNAME VARCHAR2 (10)
CUST_MNAME VARCHAR2 (10)
* CUST_LNAME VARCHAR2 (10)
* TBL_LAST_DT DATE
F * REGION_ID VARCHAR2 (4)
ASVC_CUSTOMER_PK (CUST_ID)
ASVC_CUSTOMER_ASVC_REGION_FK (REGION_ID)
ASVC_ORDR
P * ROWNUM NUMBER (15)
* ORDR_ID VARCHAR2 (15)
* ORDR_DATE DATE
* ORDR_PRIORITY VARCHAR2 (15)
* TBL_LAST_DT DATE
F * CUST_ID VARCHAR2 (15)
ASVC_ORDR_PK (ROWNUM)
ASVC_ORDR_ASVC_CUSTOMER_FK (CUST_ID)
ASVC_RETURN
P * RTRN_ID VARCHAR2 (15)
* RETURNED CHAR (1)
* TBL_LAST_DT DATE
F * ROWNUM NUMBER (15)
ASVC_RETURN_PK (RTRN_ID)
ASVC_RETURN_ASVC_ORDR_FK (ROWNUM)
ASVC_PRDCT
P * PRDCT_ID VARCHAR2 (15)
* PRDCT_NAME VARCHAR2 (15)
PRDCT_DESCP VARCHAR2 (50)
* TBL_LAST_DT DATE
F * CAT_ID VARCHAR2 (4)
ASVC_PRDCT_PK (PRDCT_ID)
ASVC_PRDCT_ASVC_CTGRY_FK (CAT_ID)
ASVC_REGION
P * REGION_ID VARCHAR2 (4)
* REGION_NAME VARCHAR2 (20)
* TBL_LAST_DT DATE
F * MARKET_ID VARCHAR2 (4)
ASVC_REGION_PK (REGION_ID)
ASVC_REGION_ASVC_MARKET_FK (MARKET_ID)
ASVC_ORDR_PRDCT
PF* PRDCT_ID VARCHAR2 (15)
PF* ROWNUM NUMBER (15)
* QTY NUMBER (4)
* SHIP_COST NUMBER (4,2)
* SHIP_DATE DATE
* DISCOUNT NUMBER (3,2)
* MODEOF_SHIP VARCHAR2 (15)
* PROFIT NUMBER (4,2)
* SALE_PRICE NUMBER (4,2)
* TBL_LAST_DT DATE
ASVC_ORDR_PRDCT_PK (PRDCT_ID, ROWNUM)
ASVC_ORDR_PRDCT_ASVC_ORDR_FK (ROWNUM)
ASVC_ORDR_PRDCT_ASVC_PRDCT_FK (PRDCT_ID)
ASVC_CTGRY
P * CAT_ID VARCHAR2 (4)
* CAT_NAME VARCHAR2 (20)
DESCP VARCHAR2 (50)
* TBL_LAST_DT DATE
ASVC_CTGRY_PK (CAT_ID)
ASVC_SUBCAT
P * SUBCAT_ID VARCHAR2 (4)
* SUBCAT_NAME VARCHAR2 (15)
DESCP VARCHAR2 (20)
* TBL_LAST_DT DATE
F * CAT_ID VARCHAR2 (4)
ASVC_SUBCAT_PK (SUBCAT_ID)
ASVC_SUBCAT_ASVC_CTGRY_FK (CAT_ID)
ASVC_MARKET
P * MARKET_ID VARCHAR2 (4)
* MARKET_NAME VARCHAR2 (15)
* TBL_LAST_DT DATE
ASVC_MARKET_PK (MARKET_ID)

DATA WAREHOUSING

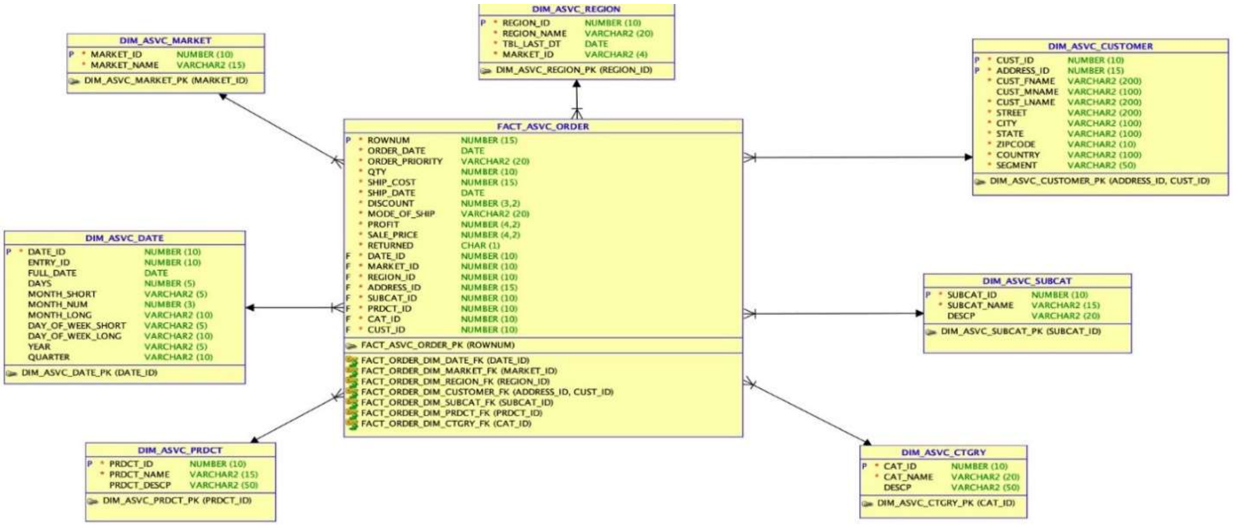
Data warehousing De-Normalizing:

Most of the modern applications need to be able to retrieve data in the shortest time possible,so that's why we de-normalize the tables to have fewer join. However, updates and inserts are more expensive, denormalization can make update and insert code harder to read, and it also might update and insert code harder to write.

OLAP LOGICAL MODEL



OLAP RELATIONAL MODEL


DIM_ASVC_MARKET
P * MARKET_ID NUMBER (10)
* MARKET_NAME VARCHAR2 (15)
DIM_ASVC_MARKET_PK (MARKET_ID)
DIM_ASVC_REGION
P * REGION_ID NUMBER (10)
* REGION_NAME VARCHAR2 (20)
* TBL_LAST_DT DATE
* MARKET_ID VARCHAR2 (4)
DIM_ASVC_REGION_PK (REGION_ID)
DIM_ASVC_CUSTOMER
P * CUST_ID NUMBER (10)
P * ADDRESS_ID NUMBER (15)
* CUST_FNAME VARCHAR2 (200)
CUST_MNAME VARCHAR2 (100)
* CUST_LNAME VARCHAR2 (200)
* STREET VARCHAR2 (200)
* CITY VARCHAR2 (100)
* STATE VARCHAR2 (100)
* ZIPCODE VARCHAR2 (10)
* COUNTRY VARCHAR2 (100)
* SEGMENT VARCHAR2 (50)
DIM_ASVC_CUSTOMER_PK (ADDRESS_ID, CUST_ID)
FACT_ASVC_ORDER
P * ROWNUM NUMBER (15)
* ORDER_DATE DATE
* ORDER_PRIORITY VARCHAR2 (20)
* QTY NUMBER (10)
* SHIP_COST NUMBER (15)
* SHIP_DATE DATE
* DISCOUNT NUMBER (3,2)
* MODE_OF_SHIP VARCHAR2 (20)
* PROFIT NUMBER (4,2)
* SALE_PRICE NUMBER (4,2)
* RETURNED CHAR (1)
F * DATE_ID NUMBER (10)
F * MARKET_ID NUMBER (10)
F * REGION_ID NUMBER (10)
F * ADDRESS_ID NUMBER (15)
F * SUBCAT_ID NUMBER (10)
F * PRDCT_ID NUMBER (10)
F * CAT_ID NUMBER (10)
F * CUST_ID NUMBER (10)
FACT_ASVC_ORDER_PK (ROWNUM)
FACT_ORDER_DIM_DATE_FK (DATE_ID)
FACT_ORDER_DIM_MARKET_FK (MARKET_ID)
FACT_ORDER_DIM_REGION_FK (REGION_ID)
FACT_ORDER_DIM_CUSTOMER_FK (ADDRESS_ID, CUST_ID)
FACT_ORDER_DIM_SUBCAT_FK (SUBCAT_ID)
FACT_ORDER_DIM_PRDCT_FK (PRDCT_ID)
FACT_ORDER_DIM_CTGRY_FK (CAT_ID)
DIM_ASVC_DATE
P * DATE_ID NUMBER (10)
ENTRY_ID NUMBER (10)
FULL_DATE DATE
DAYS NUMBER (5)
MONTH_SHORT VARCHAR2 (5)
MONTH_NUM NUMBER (3)
MONTH_LONG VARCHAR2 (10)
DAY_OF_WEEK_SHORT VARCHAR2 (5)
DAY_OF_WEEK_LONG VARCHAR2 (10)
YEAR VARCHAR2 (5)
QUARTER VARCHAR2 (10)
DIM_ASVC_DATE_PK (DATE_ID)
DIM_ASVC_SUBCAT
P * SUBCAT_ID NUMBER (10)
* SUBCAT_NAME VARCHAR2 (15)
DESCP VARCHAR2 (20)
DIM_ASVC_SUBCAT_PK (SUBCAT_ID)
DIM_ASVC_PRDCT
P * PRDCT_ID NUMBER (10)
* PRDCT_NAME VARCHAR2 (15)
PRDCT_DESCP VARCHAR2 (50)
DIM_ASVC_PRDCT_PK (PRDCT_ID)
DIM_ASVC_CTGRY
P * CAT_ID NUMBER (10)
* CAT_NAME VARCHAR2 (20)
DESCP VARCHAR2 (50)
DIM_ASVC_CTGRY_PK (CAT_ID)

ETL SUMMARY

The ETL process is a key part of the project as it enables us to extract data from the source system (an S3 bucket), transform it into a more usable format, and load it into our target system (Snowflake). The ETL process typically consists of three phases: Extract, Transform, and Load. Here's how these phases of ETL to our project:

Extract:

In this phase, we extract data from the S3 bucket and make it available for further processing. We achieved this by creating an external stage in Snowflake that is linked to the S3 bucket via an AWS S3 Integration. This stage serves as the interface between our source data and our target system (Snowflake).

Transform:

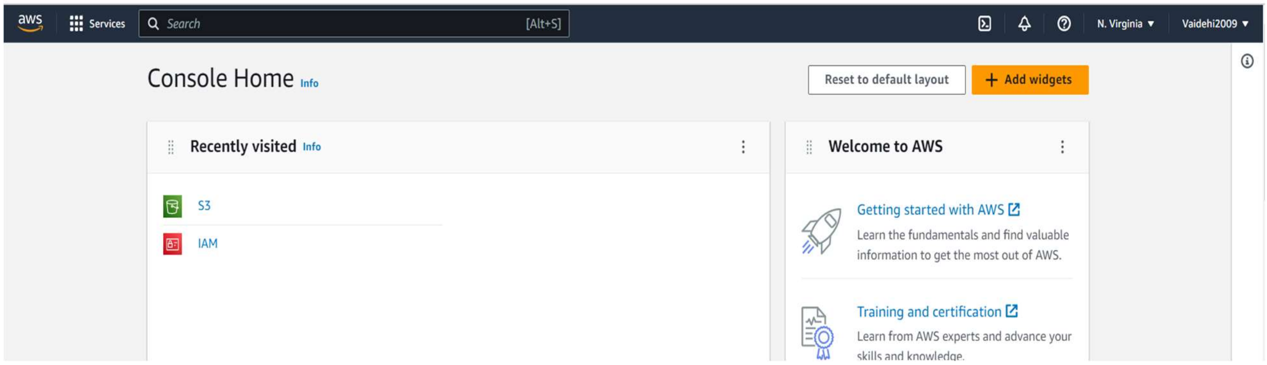
In this phase, we transform the data into a more usable format. This often involves filtering, aggregating, and joining data from different sources. In our project, we need to transform the data to conform to a specific schema or format that is required by the business needs.

Load:

In this phase, we load the transformed data into our target system (Snowflake). We accomplished this by creating a Snowflake stage that is linked to the external stage in the previous step. This stage serves as the target location in Snowflake where data will be loaded from the external S3 stage during the ETL process. Once the data is loaded, we will use it for reporting, analysis, and other business needs.

SQL AND DATA ANALYTICS FOR DATA WAREHOUSE

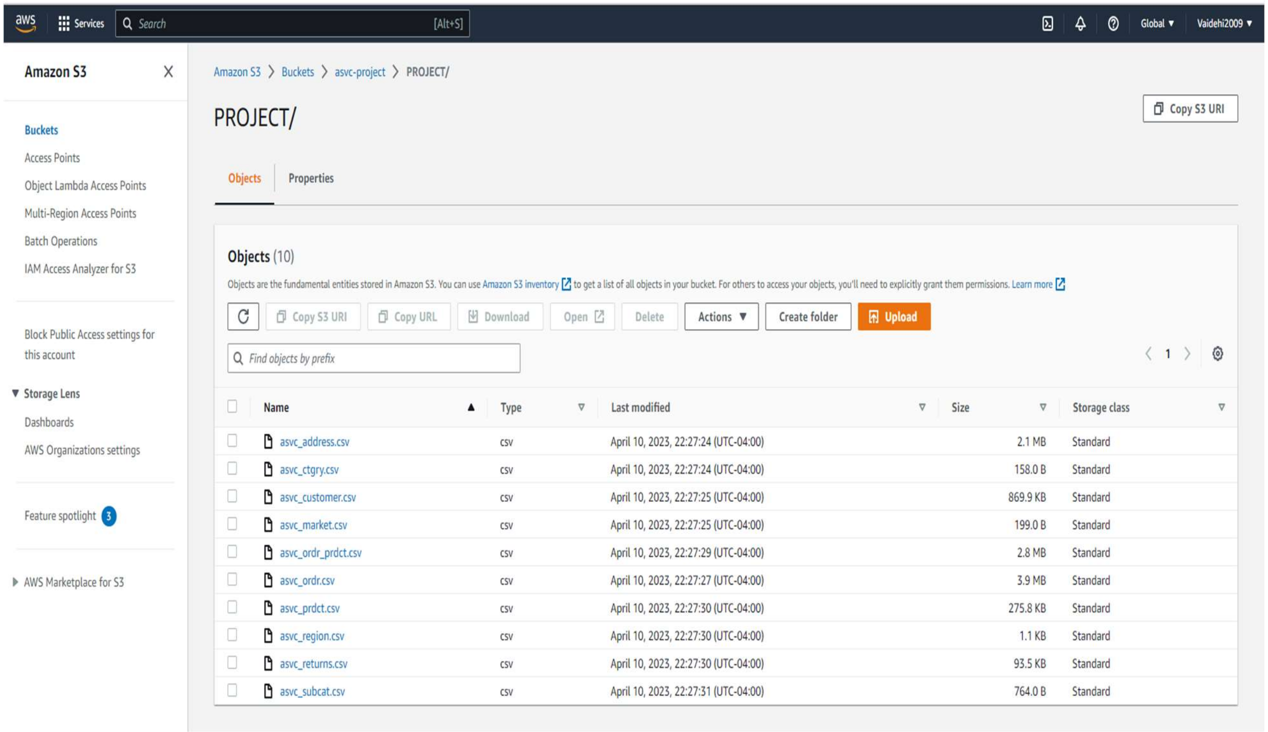
Created AWS S3 bucket and Wrote a JSON object to define an IAM policy that allows ETL tools to read from source S3 bucket, transform the data and write the results to S3 bucket.



Learn the fundamentals and find valuable information to get the most out of AWS.

Learn from AWS experts and advance your skills and knowledge.

Uploaded the csv file of OLTP to the AWS bucket:

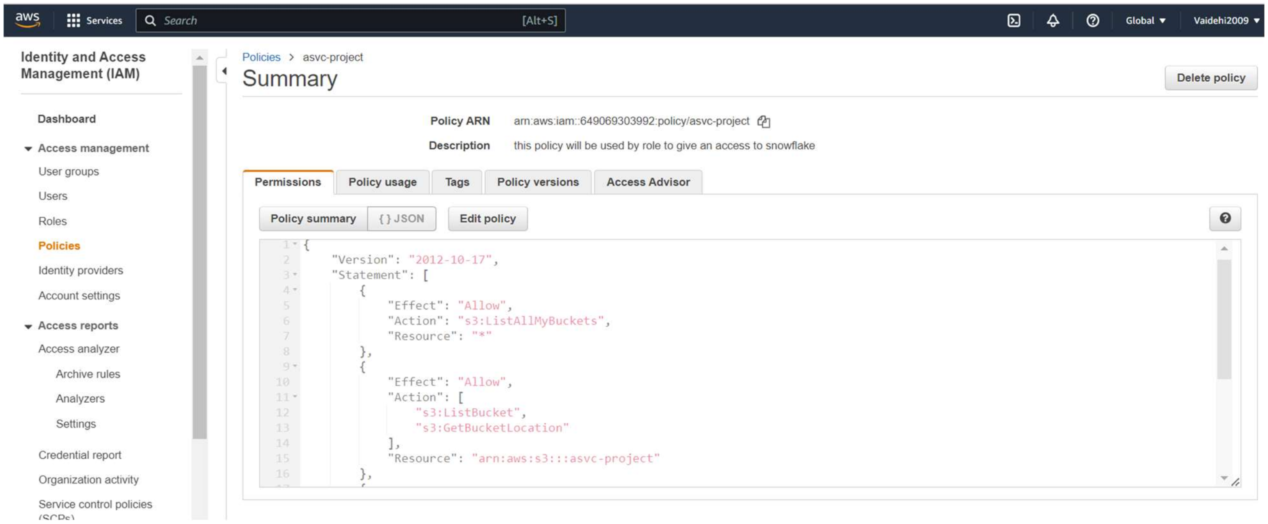


Objects (10)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
asvc_address.csv	csv	April 10, 2023, 22:27:24 (UTC-04:00)	2.1 MB	Standard
asvc_ctgry.csv	csv	April 10, 2023, 22:27:24 (UTC-04:00)	158.0 B	Standard
asvc_customer.csv	csv	April 10, 2023, 22:27:25 (UTC-04:00)	869.9 KB	Standard
asvc_market.csv	csv	April 10, 2023, 22:27:25 (UTC-04:00)	199.0 B	Standard
asvc_ordr_prdct.csv	csv	April 10, 2023, 22:27:29 (UTC-04:00)	2.8 MB	Standard
asvc_ordr.csv	csv	April 10, 2023, 22:27:27 (UTC-04:00)	3.9 MB	Standard
asvc_prdct.csv	csv	April 10, 2023, 22:27:30 (UTC-04:00)	275.8 KB	Standard
asvc_region.csv	csv	April 10, 2023, 22:27:30 (UTC-04:00)	1.1 KB	Standard
asvc_returns.csv	csv	April 10, 2023, 22:27:30 (UTC-04:00)	93.5 KB	Standard
asvc_subcat.csv	csv	April 10, 2023, 22:27:31 (UTC-04:00)	764.0 B	Standard

Create connection between AWS and snowflake, so snowflake can access the data in AWS bucket.

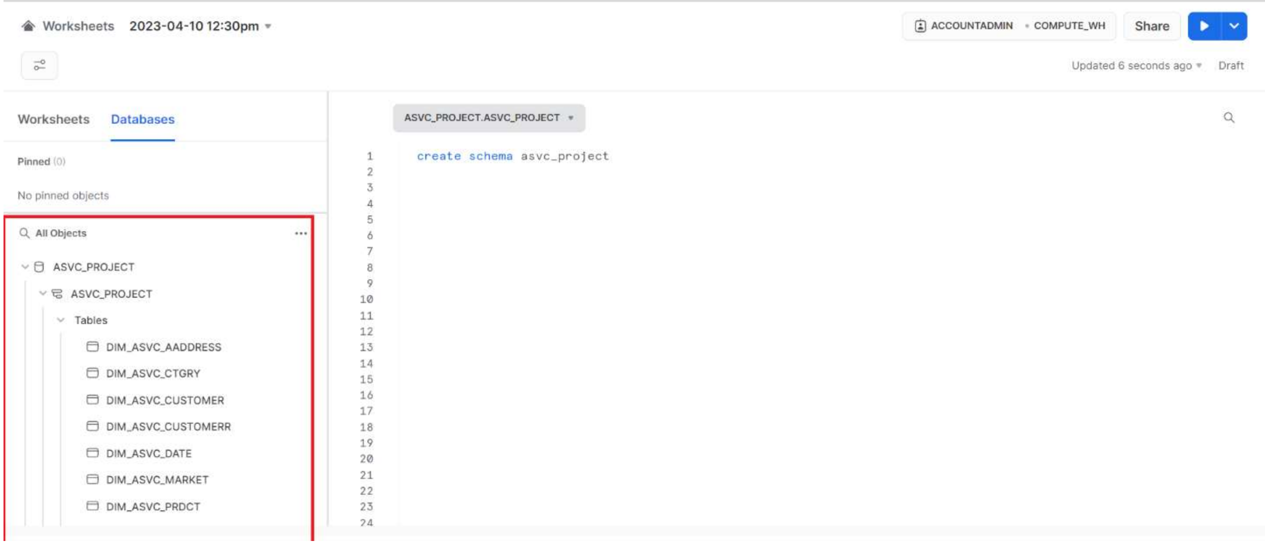


Policy ARN arn:aws:iam::649069303992:policy/asvc-project

Description this policy will be used by role to give an access to snowflake

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:ListAllMyBuckets",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation"
            ],
            "Resource": "arn:aws:s3:::asvc-project"
        },
```

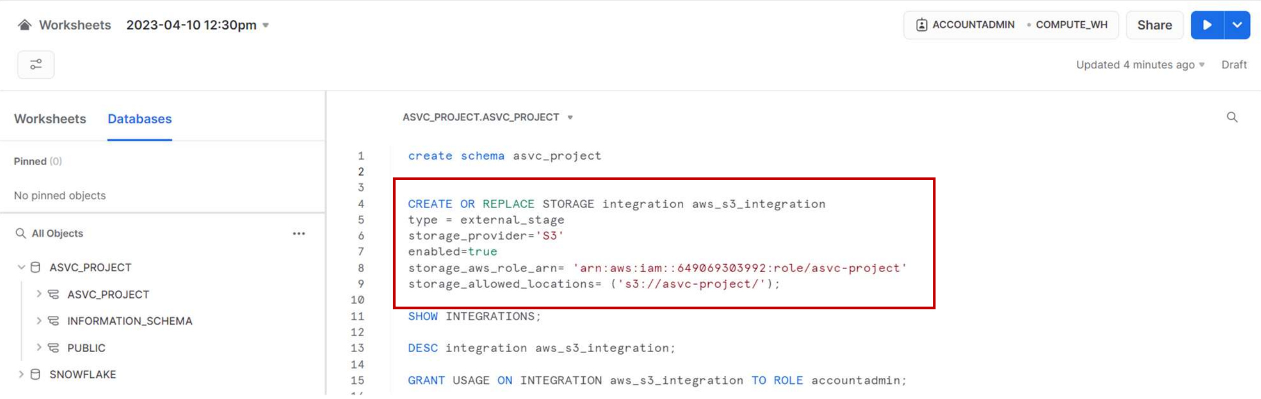
Generated de-normalized OLAP for data warehouse in which ETL process will be performed and the data from OLTP will be loaded.



```
create schema asvc_project
```

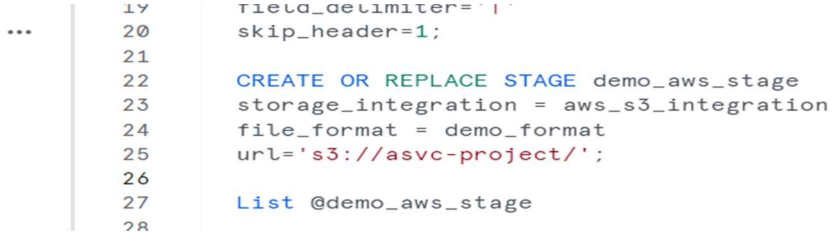
This is a key step in the ETL process, specifically the "Extract" phase, where data is extracted from the source system (in this case, an S3 bucket) and made available for further processing.

Created a storage integration, this sets up the integration between Snowflake and the S3 bucket, allowing Snowflake to access and load data from the external stage.



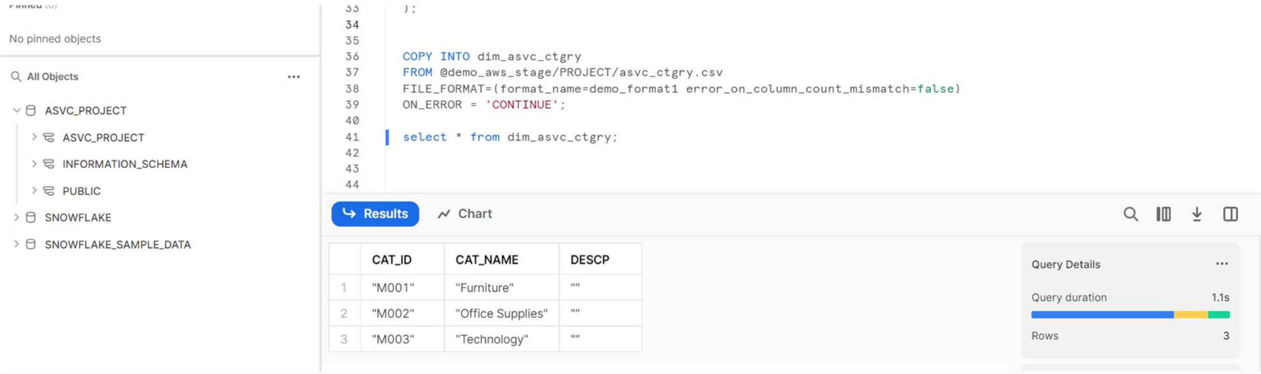
```
create schema asvc_project


CREATE OR REPLACE STORAGE integration aws_s3_integration
type = external_stage
storage_provider='S3'
enabled=true
storage_aws_role_arn= 'arn:aws:iam::649069303992:role/asvc-project'
storage_allowed_locations= ('s3://asvc-project/');

SHOW INTEGRATIONS;

DESC integration aws_s3_integration;

GRANT USAGE ON INTEGRATION aws_s3_integration TO ROLE accountadmin;
```

Created a Snowflake Stage which sets up a target location in Snowflake where data will be loaded from the external S3 stage during the ETL process:



```
field_delimiter='|'
skip_header=1;

CREATE OR REPLACE STAGE demo_aws_stage
storage_integration = aws_s3_integration
file_format = demo_format
url='s3://asvc-project/';

List @demo_aws_stage
```

Loading of the unstructured data from CSV file to the OLAP table dim_asvc_category:



```
);



COPY INTO dim_asvc_ctgry
FROM @demo_aws_stage/PROJECT/asvc_ctgry.csv
FILE_FORMAT=(format_name=demo_format1 error_on_column_count_mismatch=false)
ON_ERROR = 'CONTINUE';

select * from dim_asvc_ctgry;
```

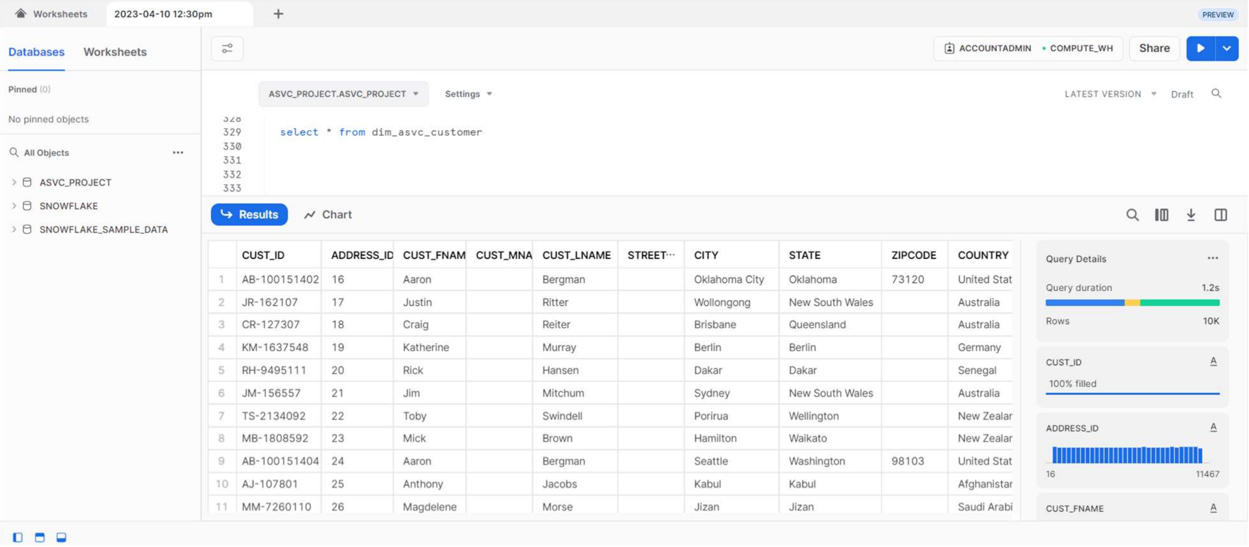
	CAT_ID	CAT_NAME	DESCP
1	"M001"	"Furniture"	""
2	"M002"	"Office Supplies"	""
3	"M003"	"Technology"	""

Query Details

Query duration 1.1s

Rows 3

Loading of the unstructured data from CSV file to the OLAP table dim_asvc_customer:



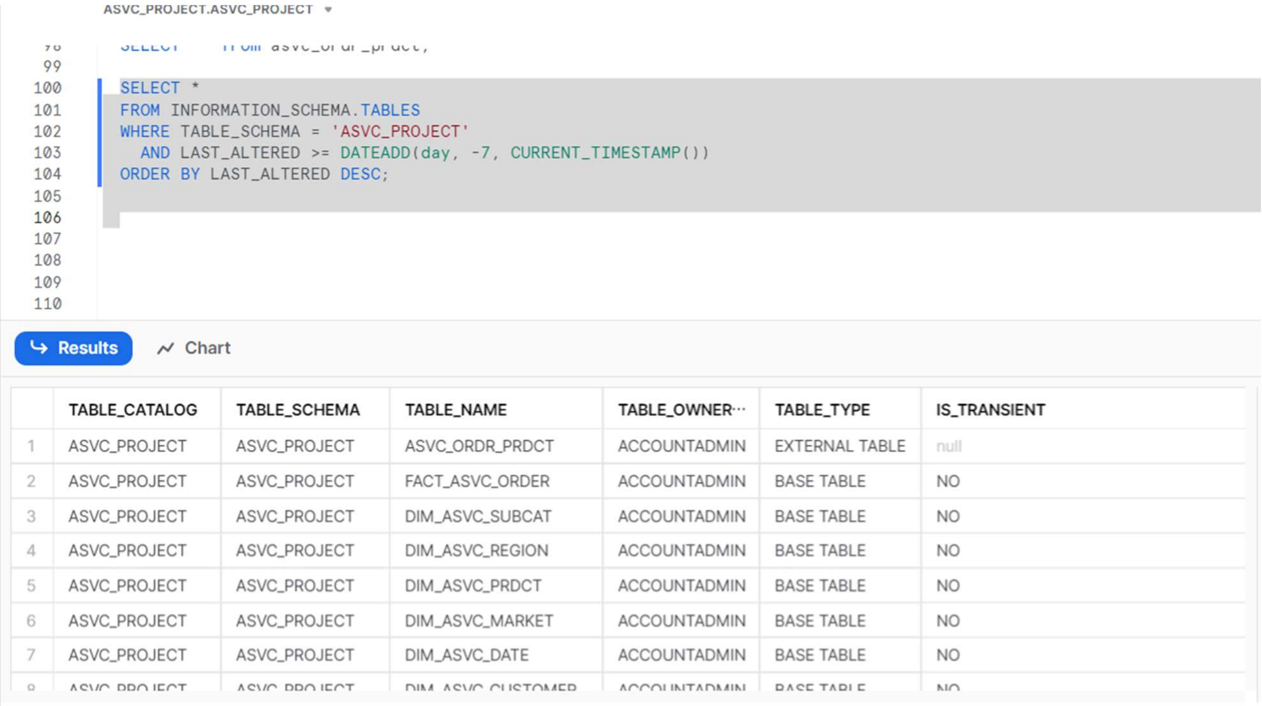
```
select * from dim_asvc_customer
```

	CUST_ID	ADDRESS_ID	CUST_FNAM	CUST_MNA	CUST_LNAME	STREET···	CITY	STATE	ZIPCODE	COUNTRY
1	AB-100151402	16	Aaron		Bergman		Oklahoma City	Oklahoma	73120	United Stat
2	JR-162107	17	Justin		Ritter		Wollongong	New South Wales		Australia
3	CR-127307	18	Craig		Reiter		Brisbane	Queensland		Australia
4	KM-1637548	19	Katherine		Murray		Berlin	Berlin		Germany
5	RH-9495111	20	Rick		Hansen		Dakar	Dakar		Senegal
6	JM-156557	21	Jim		Mitchum		Sydney	New South Wales		Australia
7	TS-2134092	22	Toby		Swindell		Porirua	Wellington		New Zealar
8	MB-1808592	23	Mick		Brown		Hamilton	Waikato		New Zealar
9	AB-100151404	24	Aaron		Bergman		Seattle	Washington	98103	United Stat
10	AJ-107801	25	Anthony		Jacobs		Kabul	Kabul		Afghanistar
11	MM-7260110	26	Magdelene		Morse		Jizan	Jizan		Saudi Arabi

Query duration 1.2s

Rows 10K

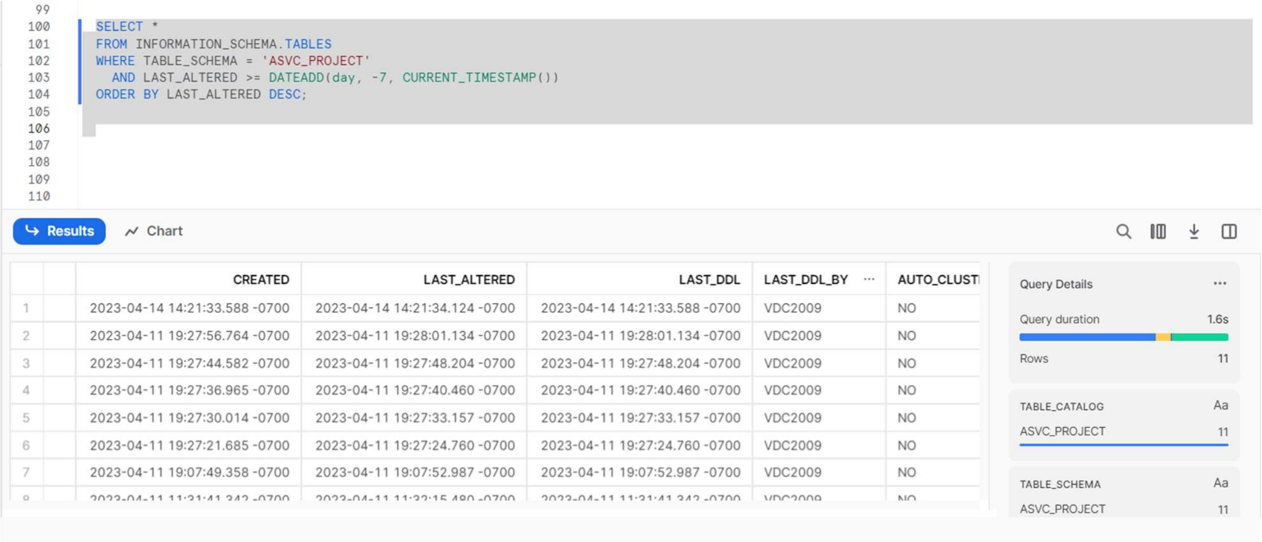
Retrieved information about all tables in the specified schema that have been altered within the last 7 days, sorted in descending order based on the date they were last altered.



```
SELECT * from asvc_ordr_prdct;

SELECT *
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'ASVC_PROJECT'
  AND LAST_ALTERED >= DATEADD(day, -7, CURRENT_TIMESTAMP())
ORDER BY LAST_ALTERED DESC;
```

	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_OWNER···	TABLE_TYPE	IS_TRANSIENT
1	ASVC_PROJECT	ASVC_PROJECT	ASVC_ORDR_PRDCT	ACCOUNTADMIN	EXTERNAL TABLE	null
2	ASVC_PROJECT	ASVC_PROJECT	FACT_ASVC_ORDER	ACCOUNTADMIN	BASE TABLE	NO
3	ASVC_PROJECT	ASVC_PROJECT	DIM_ASVC_SUBCAT	ACCOUNTADMIN	BASE TABLE	NO
4	ASVC_PROJECT	ASVC_PROJECT	DIM_ASVC_REGION	ACCOUNTADMIN	BASE TABLE	NO
5	ASVC_PROJECT	ASVC_PROJECT	DIM_ASVC_PRDCT	ACCOUNTADMIN	BASE TABLE	NO
6	ASVC_PROJECT	ASVC_PROJECT	DIM_ASVC_MARKET	ACCOUNTADMIN	BASE TABLE	NO
7	ASVC_PROJECT	ASVC_PROJECT	DIM_ASVC_DATE	ACCOUNTADMIN	BASE TABLE	NO
8	ASVC_PROJECT	ASVC_PROJECT	DIM_ASVC_CUSTOMER	ACCOUNTADMIN	BASE TABLE	NO

Continuation



```
SELECT *
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'ASVC_PROJECT'
  AND LAST_ALTERED >= DATEADD(day, -7, CURRENT_TIMESTAMP())
ORDER BY LAST_ALTERED DESC;
```

		CREATED	LAST_ALTERED	LAST_DDL	LAST_DDL_BY	AUTO_CLUSTI
1		2023-04-14 14:21:33.588 -0700	2023-04-14 14:21:34.124 -0700	2023-04-14 14:21:33.588 -0700	VDC2009	NO
2		2023-04-11 19:27:56.764 -0700	2023-04-11 19:28:01.134 -0700	2023-04-11 19:28:01.134 -0700	VDC2009	NO
3		2023-04-11 19:27:44.582 -0700	2023-04-11 19:27:48.204 -0700	2023-04-11 19:27:48.204 -0700	VDC2009	NO
4		2023-04-11 19:27:36.965 -0700	2023-04-11 19:27:40.460 -0700	2023-04-11 19:27:40.460 -0700	VDC2009	NO
5		2023-04-11 19:27:30.014 -0700	2023-04-11 19:27:33.157 -0700	2023-04-11 19:27:33.157 -0700	VDC2009	NO
6		2023-04-11 19:27:21.685 -0700	2023-04-11 19:27:24.760 -0700	2023-04-11 19:27:24.760 -0700	VDC2009	NO
7		2023-04-11 19:07:49.358 -0700	2023-04-11 19:07:52.987 -0700	2023-04-11 19:07:52.987 -0700	VDC2009	NO
8		2023-04-11 11:31:41.342 -0700	2023-04-11 11:32:15.480 -0700	2023-04-11 11:31:41.342 -0700	VDC2009	NO

Query Details

Query duration 1.6s

Rows 11

TABLE_CATALOG Aa

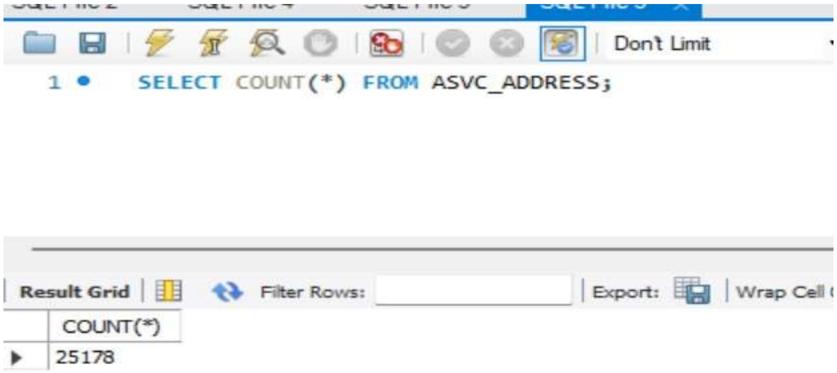
ASVC_PROJECT 11

TABLE_SCHEMA Aa

ASVC_PROJECT 11

RECORD COUNT OF ALL TABLES IN OLTP

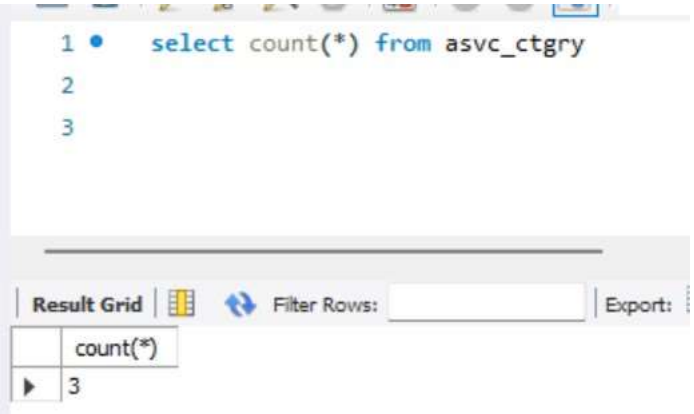
Count for asvc_address:



```
SELECT COUNT(*) FROM ASVC_ADDRESS;
```

```
COUNT(*)
25178
```

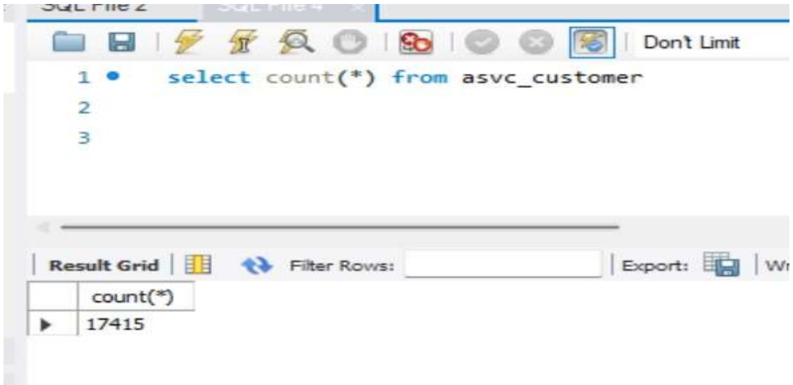
Count for asvc_ctgry:



```
select count(*) from asvc_ctgry
```

```
count(*)
3
```

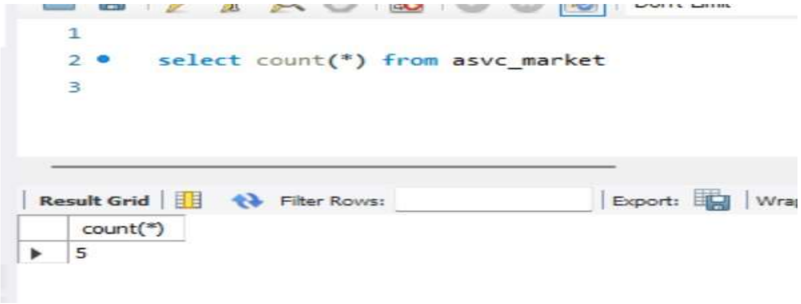
Count for asvc_customer:



```
select count(*) from asvc_customer
```

```
count(*)
17415
```

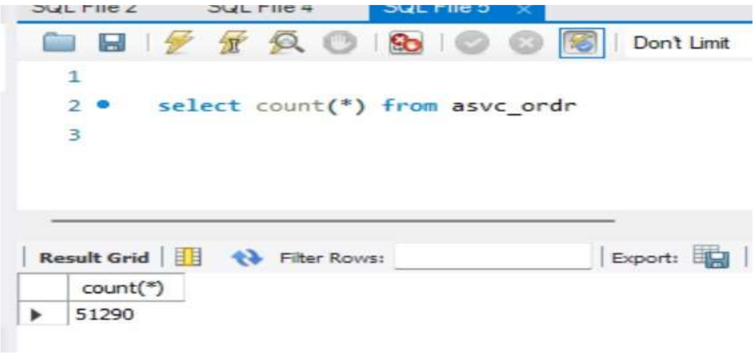
Count for asvc_market:



```
select count(*) from asvc_market
```

```
count(*)
5
```

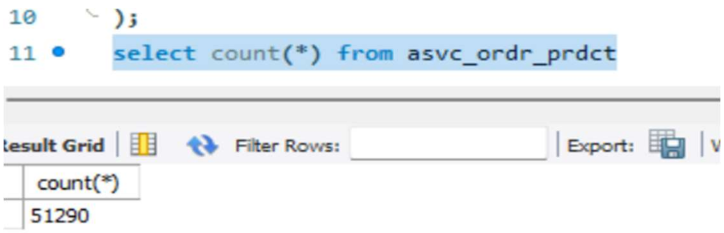
Count for asvc_ordr:



```
select count(*) from asvc_ordr
```

```
count(*)
51290
```

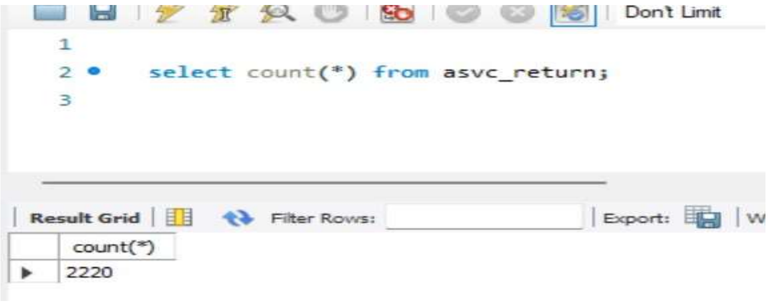
Count for asvc_ordr_prdct:



```
);
select count(*) from asvc_ordr_prdct
```

```
count(*)
51290
```

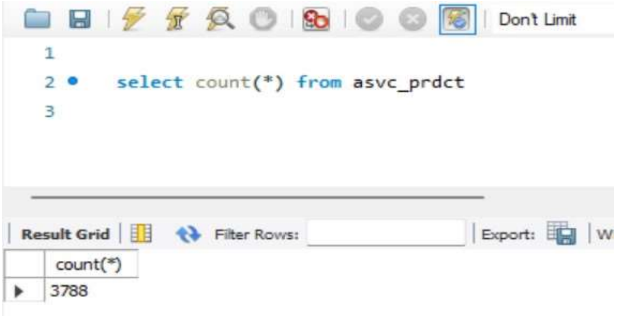
Count for asvc_return:



```
select count(*) from asvc_return;
```

```
count(*)
2220
```

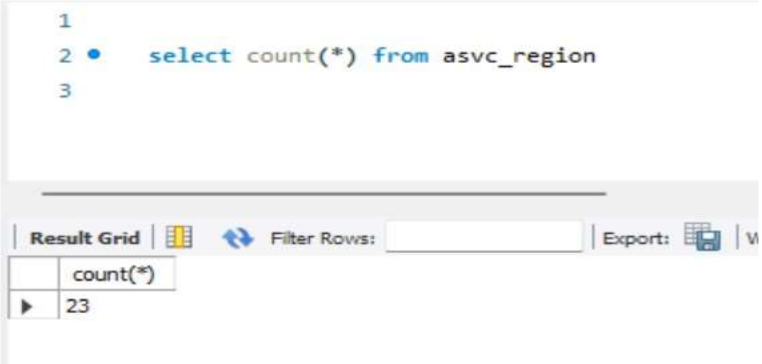
Count for asvc_prdct:



```
select count(*) from asvc_prdct
```

```
count(*)
3788
```

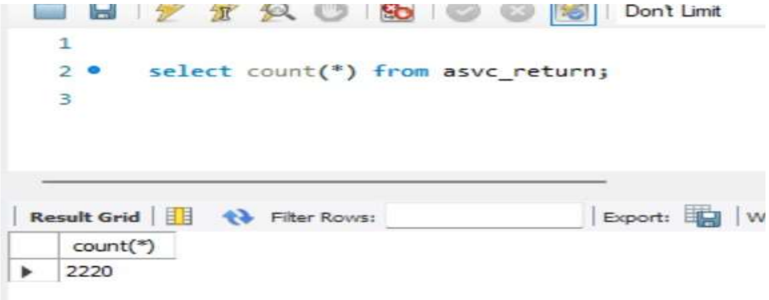
Count for asvc_region:



```
select count(*) from asvc_region
```

```
count(*)
23
```

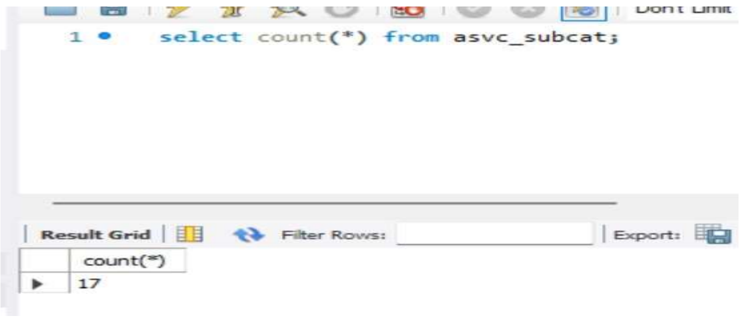
Count for asvc_return:



```
select count(*) from asvc_return;
```

```
count(*)
2220
```

Count for asvc_subcat:



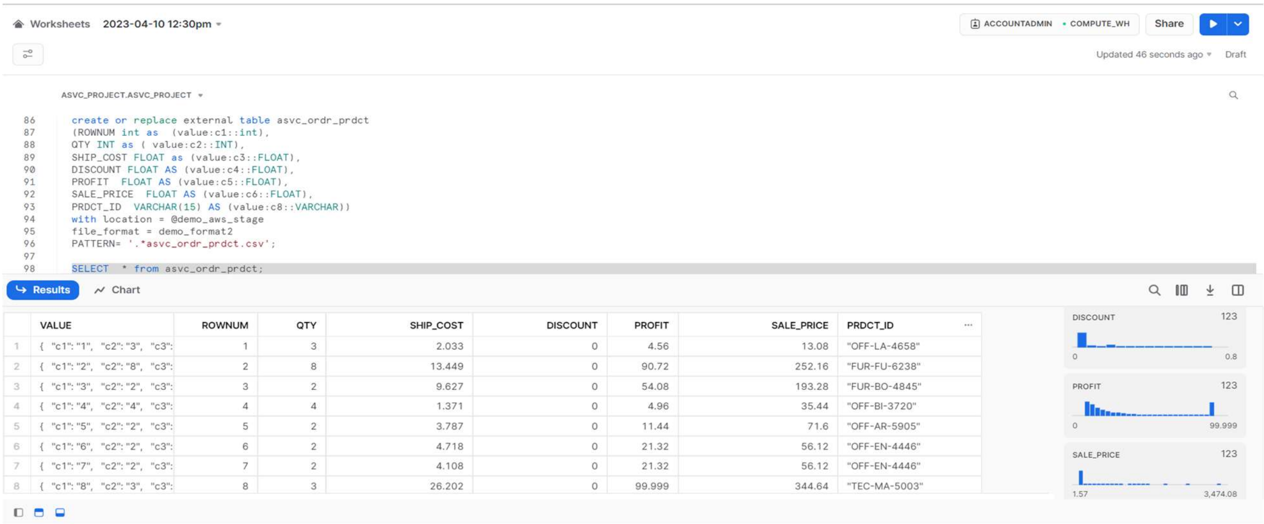
```
select count(*) from asvc_subcat;
```

```
count(*)
17
```

DATA MANAGEMENT

EXTERNAL TABLE:

Created an **“External table”** that query the data in the CSV files stored in S3 as if they were regular Snowflake tables, without the need to load the data into Snowflake first.

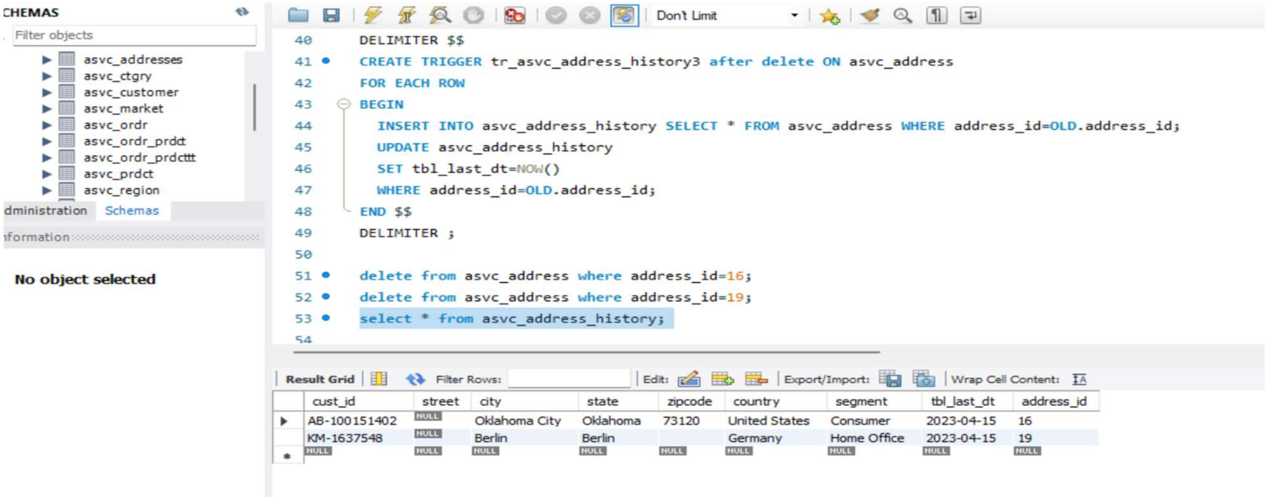


```
create or replace external table asvc_ordr_prdct
(ROWNUM int as  (value:c1::int),
QTY INT as ( value:c2::INT),
SHIP_COST FLOAT as (value:c3::FLOAT),
DISCOUNT FLOAT AS (value:c4::FLOAT),
PROFIT  FLOAT AS (value:c5::FLOAT),
SALE_PRICE  FLOAT AS (value:c6::FLOAT),
PRDCT_ID  VARCHAR(15) AS (value:c8::VARCHAR))
with location = @demo_aws_stage
file_format = demo_format2
PATTERN= '.*asvc_ordr_prdct.csv';

SELECT  * from asvc_ordr_prdct;
```

	VALUE	ROWNUM	QTY	SHIP_COST	DISCOUNT	PROFIT	SALE_PRICE	PRDCT_ID
1	{ "c1": "1", "c2": "3", "c3":	1	3	2.033	0	4.56	13.08	"OFF-LA-4658"
2	{ "c1": "2", "c2": "8", "c3":	2	8	13.449	0	90.72	252.16	"FUR-FU-6238"
3	{ "c1": "3", "c2": "2", "c3":	3	2	9.627	0	54.08	193.28	"FUR-BO-4845"
4	{ "c1": "4", "c2": "4", "c3":	4	4	1.371	0	4.96	35.44	"OFF-BI-3720"
5	{ "c1": "5", "c2": "2", "c3":	5	2	3.787	0	11.44	71.6	"OFF-AR-5905"
6	{ "c1": "6", "c2": "2", "c3":	6	2	4.718	0	21.32	56.12	"OFF-EN-4446"
7	{ "c1": "7", "c2": "2", "c3":	7	2	4.108	0	21.32	56.12	"OFF-EN-4446"
8	{ "c1": "8", "c2": "3", "c3":	8	3	26.202	0	99.999	344.64	"TEC-MA-5003"

HISTORY TABLE:

Created history table in OLTP for asvc_address where all the deleted records will be stored in history table.



```
DELIMITER $$
CREATE TRIGGER tr_asvc_address_history3 after delete ON asvc_address
FOR EACH ROW
BEGIN
  INSERT INTO asvc_address_history SELECT * FROM asvc_address WHERE address_id=OLD.address_id;
  UPDATE asvc_address_history
  SET tbl_last_dt=NOW()
  WHERE address_id=OLD.address_id;
END $$
DELIMITER ;

delete from asvc_address where address_id=16;
delete from asvc_address where address_id=19;
select * from asvc_address_history;
```

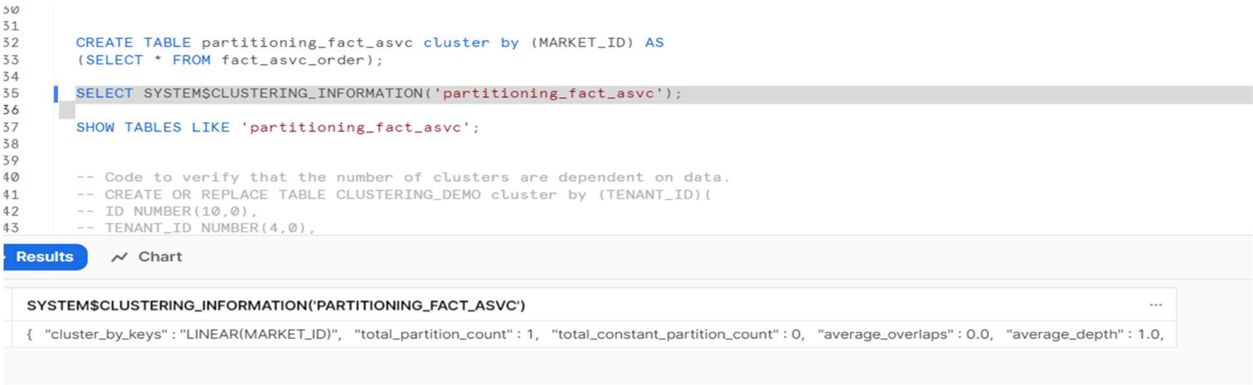
	cust_id	street	city	state	zipcode	country	segment	tbl_last_dt	address_id
▶	AB-100151402	NULL	Oklahoma City	Oklahoma	73120	United States	Consumer	2023-04-15	16
	KM-1637548	NULL	Berlin	Berlin		Germany	Home Office	2023-04-15	19
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

PARTITION TABLE

In Snowflake, table partitioning is not supported in the traditional sense, but clustering is a feature that serves a similar purpose. Table clustering is the process of organizing data within a table based on the values in one or more columns. It reorders the data in the table so that rows with similar values are stored together physically, allowing for more efficient data retrieval.

When a table is clustered on one or more columns, Snowflake uses those columns to group together data in the same micro-partitions. By clustering on specific columns, Snowflake can ensure that rows with similar values will be stored in the same micro-partitions, which can make queries more efficient by minimizing the amount of data that needs to be scanned.

We have used clustering which serves the same purpose as table partitioning



```


CREATE TABLE partitioning_fact_asvc cluster by (MARKET_ID) AS
(SELECT * FROM fact_asvc_order);

SELECT SYSTEM$CLUSTERING_INFORMATION('partitioning_fact_asvc');

SHOW TABLES LIKE 'partitioning_fact_asvc';


-- Code to verify that the number of clusters are dependent on data.
-- CREATE OR REPLACE TABLE CLUSTERING_DEMO cluster by (TENANT_ID)(
-- ID NUMBER(10,0),
-- TENANT_ID NUMBER(4,0),
```

```
SYSTEM$CLUSTERING_INFORMATION('PARTITIONING_FACT_ASVC')
{  "cluster_by_keys" : "LINEAR(MARKET_ID)",  "total_partition_count" : 1,  "total_constant_partition_count" : 0,  "average_overlaps" : 0.0,  "average_depth" : 1.0,
```

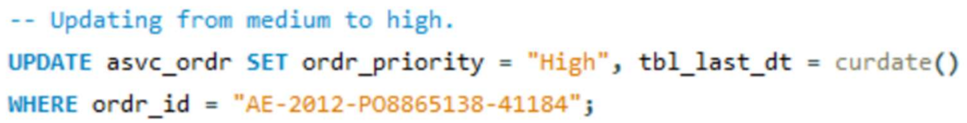
In our project, we have set the Market ID as the partition ID because we have only five markets, which will essentially create five clusters of data. This will enable us to search for specific data in a particular cluster according to the market. However, Snowflake's internal implementation of partitioning is dependent on the size of our data and available nodes.

Since we have around 50k rows, it is not sufficient for Snowflake to automatically create partitions. Therefore, the partition count is still coming out to be one. However, Snowflake will automatically make partitions in the future if the data scales to a larger extent. To verify this, we have created a dummy table and filled it with random data, and we found that Snowflake created partitions when the data exceeded a certain threshold.

INCREMENTAL ETL

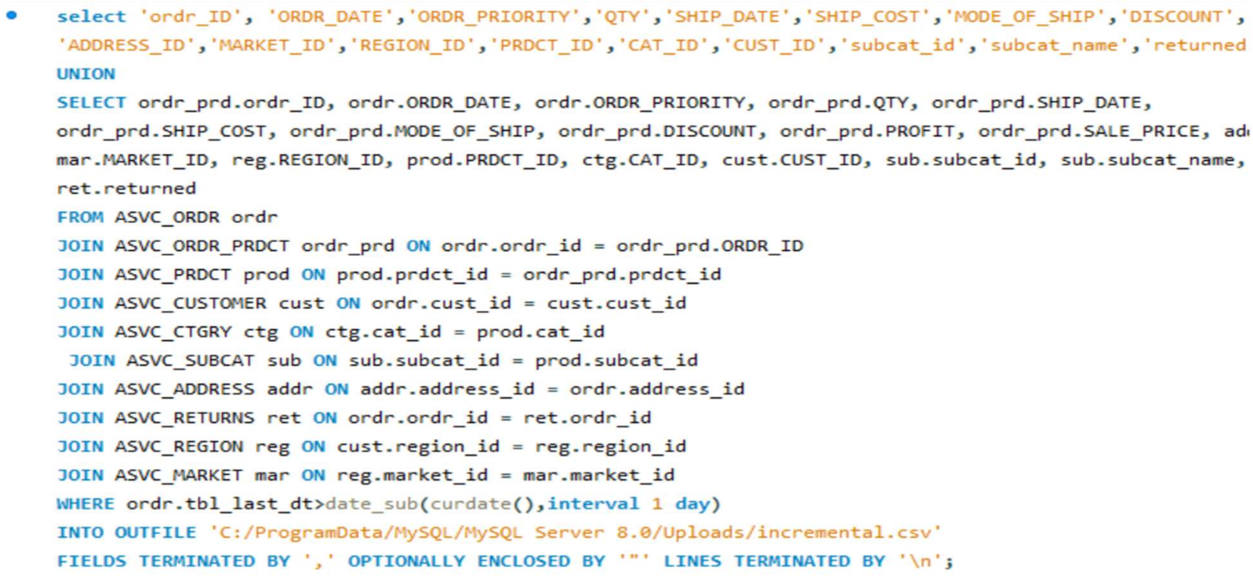
Incremental ETL is a method of data extraction and processing that involves extracting only the data that has changed since the last ETL run, transforms it into the desired format, and loads it into the target system, rather than processing the entire data set each time. This method is often used to reduce processing time and improve efficiency, especially for large and constantly changing data sets.

For demo, we are changing the order_priority from medium to high for a particular order_id in order to update these changes in the Snowflake schema.



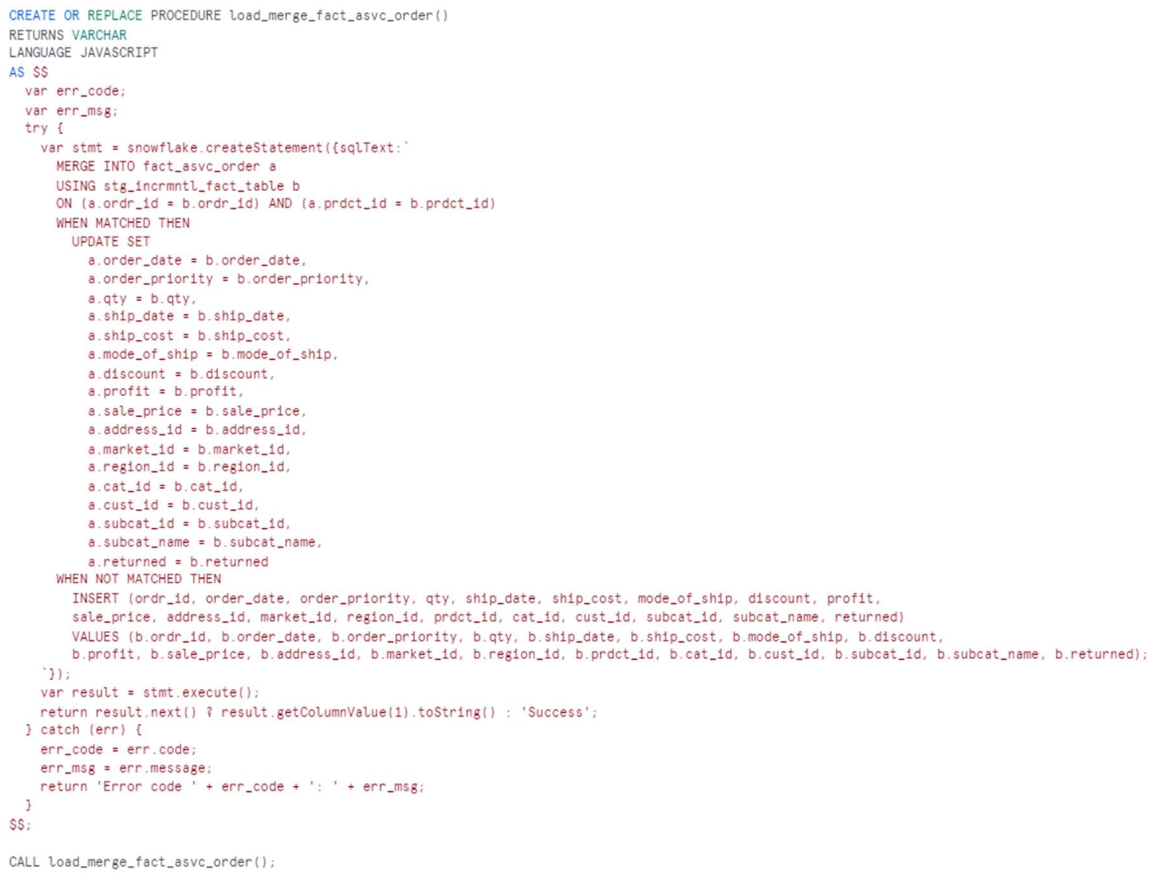
```
-- Updating from medium to high.
UPDATE asvc_ordr SET ordr_priority = "High", tbl_last_dt = curdate()
WHERE ordr_id = "AE-2012-PO8865138-41184";
```

This query selects records modified in the last 24 hours. The results are written to a CSV file, which is then pushed in AWS S3 bucket.



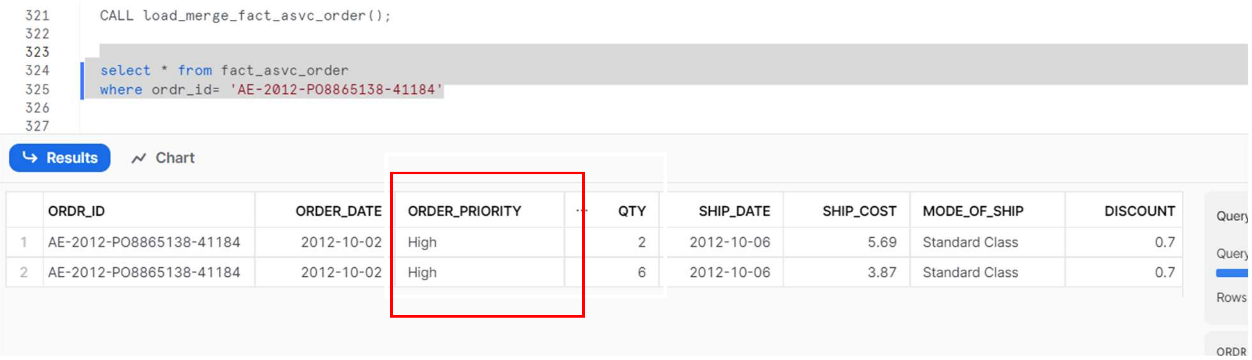
```
•    select 'ordr_ID', 'ORDR_DATE','ORDR_PRIORITY','QTY','SHIP_DATE','SHIP_COST','MODE_OF_SHIP','DISCOUNT',
     'ADDRESS_ID','MARKET_ID','REGION_ID','PRDCT_ID','CAT_ID','CUST_ID','subcat_id','subcat_name','returned
     UNION
     SELECT ordr_prd.ordr_ID, ordr.ORDR_DATE, ordr.ORDR_PRIORITY, ordr_prd.QTY, ordr_prd.SHIP_DATE,
     ordr_prd.SHIP_COST, ordr_prd.MODE_OF_SHIP, ordr_prd.DISCOUNT, ordr_prd.PROFIT, ordr_prd.SALE_PRICE, ad
     mar.MARKET_ID, reg.REGION_ID, prod.PRDCT_ID, ctg.CAT_ID, cust.CUST_ID, sub.subcat_id, sub.subcat_name,
     ret.returned
     FROM ASVC_ORDR ordr
     JOIN ASVC_ORDR_PRDCT ordr_prd ON ordr.ordr_id = ordr_prd.ORDR_ID
     JOIN ASVC_PRDCT prod ON prod.prdct_id = ordr_prd.prdct_id
     JOIN ASVC_CUSTOMER cust ON ordr.cust_id = cust.cust_id
     JOIN ASVC_CTGRY ctg ON ctg.cat_id = prod.cat_id
      JOIN ASVC_SUBCAT sub ON sub.subcat_id = prod.subcat_id
     JOIN ASVC_ADDRESS addr ON addr.address_id = ordr.address_id
     JOIN ASVC_RETURNS ret ON ordr.ordr_id = ret.ordr_id
     JOIN ASVC_REGION reg ON cust.region_id = reg.region_id
     JOIN ASVC_MARKET mar ON reg.market_id = mar.market_id
     WHERE ordr.tbl_last_dt>date_sub(curdate(),interval 1 day)
     INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/incremental.csv'
     FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' LINES TERMINATED BY '\n';
```

After uploading the CSV file in AWS S3 bucket, we wrote a procedure which will update the new data into the existing fact_table.



```
CREATE OR REPLACE PROCEDURE load_merge_fact_asvc_order()
RETURNS VARCHAR
LANGUAGE JAVASCRIPT
AS $$
  var err_code;
  var err_msg;
  try {
    var stmt = snowflake.createStatement({sqlText:`
      MERGE INTO fact_asvc_order a
      USING stg_incrmntl_fact_table b
      ON (a.ordr_id = b.ordr_id) AND (a.prdct_id = b.prdct_id)
      WHEN MATCHED THEN
        UPDATE SET
          a.order_date = b.order_date,
          a.order_priority = b.order_priority,
          a.qty = b.qty,
          a.ship_date = b.ship_date,
          a.ship_cost = b.ship_cost,
          a.mode_of_ship = b.mode_of_ship,
          a.discount = b.discount,
          a.profit = b.profit,
          a.sale_price = b.sale_price,
          a.address_id = b.address_id,
          a.market_id = b.market_id,
          a.region_id = b.region_id,
          a.cat_id = b.cat_id,
          a.cust_id = b.cust_id,
          a.subcat_id = b.subcat_id,
          a.subcat_name = b.subcat_name,
          a.returned = b.returned
      WHEN NOT MATCHED THEN
        INSERT (ordr_id, order_date, order_priority, qty, ship_date, ship_cost, mode_of_ship, discount, profit,
        sale_price, address_id, market_id, region_id, prdct_id, cat_id, cust_id, subcat_id, subcat_name, returned)
        VALUES (b.ordr_id, b.order_date, b.order_priority, b.qty, b.ship_date, b.ship_cost, b.mode_of_ship, b.discount,
        b.profit, b.sale_price, b.address_id, b.market_id, b.region_id, b.prdct_id, b.cat_id, b.cust_id, b.subcat_id, b.subcat_name, b.returned);
    `});
    var result = stmt.execute();
    return result.next() ? result.getColumnValue(1).toString() : 'Success';
  } catch (err) {
    err_code = err.code;
    err_msg = err.message;
    return 'Error code ' + err_code + ': ' + err_msg;
  }
$$;

CALL load_merge_fact_asvc_order();
```

Now as we can see in the below screenshot that the order_priority has changed from medium to high.



```
CALL load_merge_fact_asvc_order();


select * from fact_asvc_order
where ordr_id= 'AE-2012-PO8865138-41184'
```

	ORDR_ID	ORDER_DATE	ORDER_PRIORITY	QTY	SHIP_DATE	SHIP_COST	MODE_OF_SHIP	DISCOUNT
1	AE-2012-PO8865138-41184	2012-10-02	High	2	2012-10-06	5.69	Standard Class	0.7
2	AE-2012-PO8865138-41184	2012-10-02	High	6	2012-10-06	3.87	Standard Class	0.7

DATA VISUALIZATION

Data Visualization is a powerful tool that can help you convey complex information in a clear and efficient manner. By using graphics such as charts, tables, and infographics, you can make large amounts of data more accessible and understandable. With the help of visual analytics and dashboards, you can analyze raw data and gain insights into your business operations, both past and future.

By sharing your findings and monitoring progress, you can make better strategic decisions and identify new opportunities. Overall, data visualization can greatly enhance your ability to understand and communicate important information.

We have used Tableau Desktop 2023.1 as a Data visualization tool for creating charts and graphs as it is easyto use and low barrier to entry.

We created a link between Snowflake and Tableau so that Tableau can directly access the data and the schema that is present in the Snowflake data warehouse.

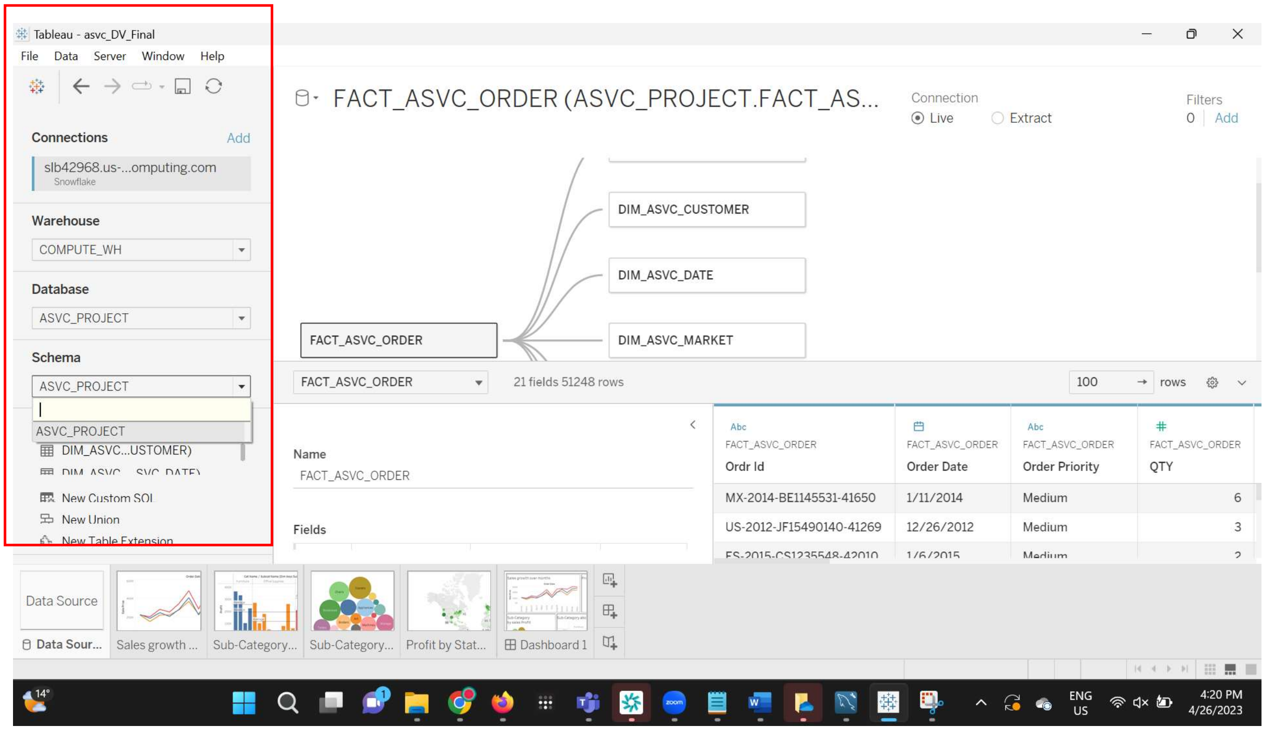


Tableau - asvc_DV_Final

Connections: slb42968.us-...omputing.com (Snowflake)

Warehouse: COMPUTE_WH

Database: ASVC_PROJECT

Schema: ASVC_PROJECT

FACT_ASVC_ORDER (ASVC_PROJECT.FACT_AS...

Connection: ● Live ○ Extract

Filters: 0

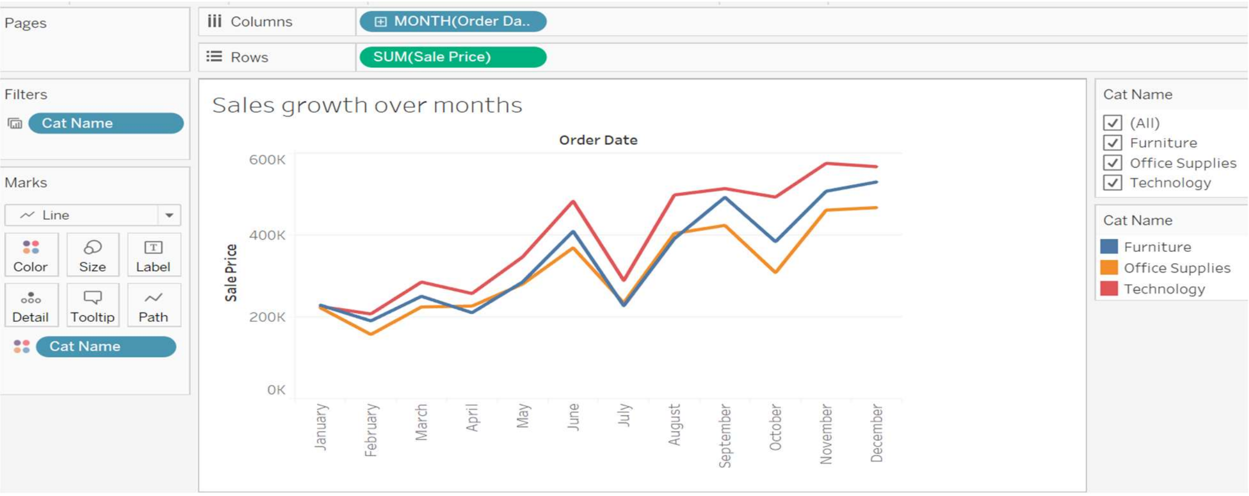
FACT_ASVC_ORDER — 21 fields 51248 rows — 100 rows

Name: FACT_ASVC_ORDER

FACT_ASVC_ORDER Ordr Id	FACT_ASVC_ORDER Order Date	FACT_ASVC_ORDER Order Priority	FACT_ASVC_ORDER QTY
MX-2014-BE1145531-41650	1/11/2014	Medium	6
US-2012-JF15490140-41269	12/26/2012	Medium	3
ES-2015-CS1235548-42010	1/6/2015	Medium	2

Worksheet 1: Sales growth over months for the category.

Sales growth over months for the category Furniture, office supplies, and technology. This graph is necessary because we can quickly see how sales are changing over time, which categories are performing better or worse, and where there might be opportunities to improve performance.


Pages

Columns: MONTH(Order Da..

Rows: SUM(Sale Price)

Filters: Cat Name

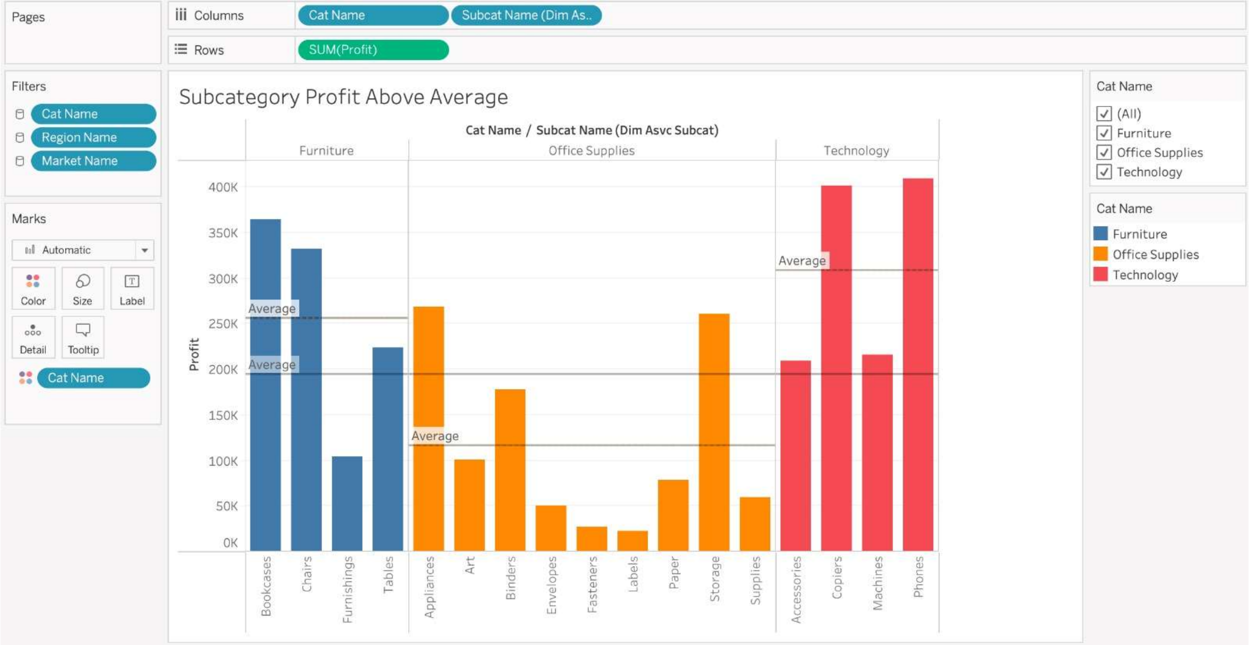
Marks: Line

Cat Name

Cat Name
☑ (All)
☑ Furniture
☑ Office Supplies
☑ Technology

Worksheet 2: Sub-Category Profit Above average

This type of graph is particularly useful because it allows us to quickly and easily identify which sub-categories are performing above or below average.


Pages

Columns: Cat Name, Subcat Name (Dim As..

Rows: SUM(Profit)

Filters: Cat Name, Region Name, Market Name

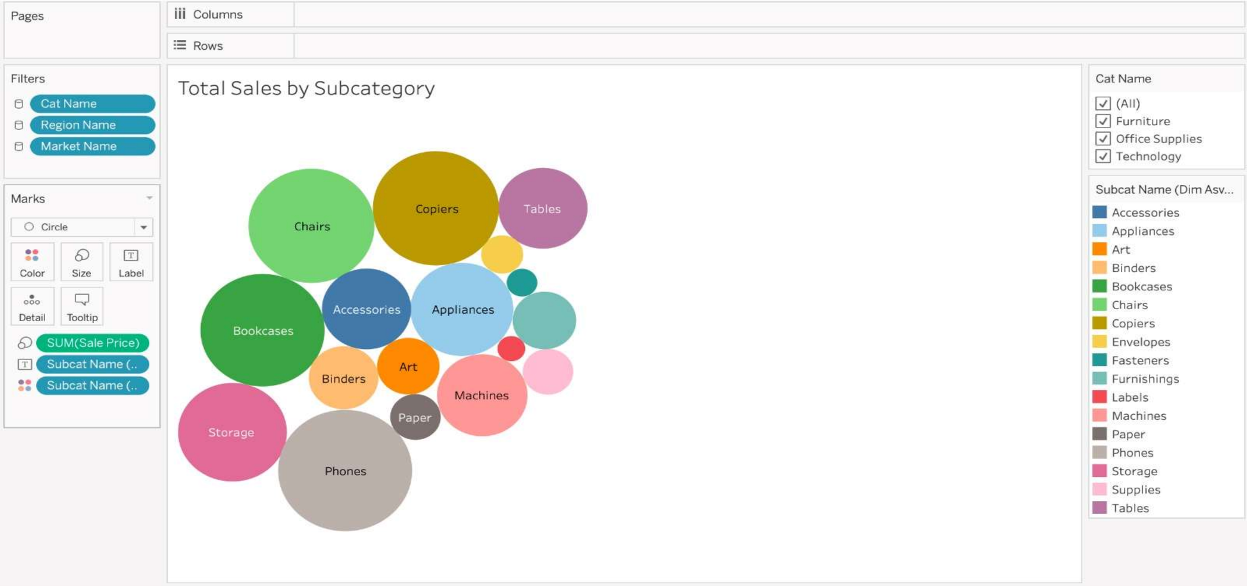
Marks: Automatic

Cat Name

Cat Name
☑ (All)
☑ Furniture
☑ Office Supplies
☑ Technology

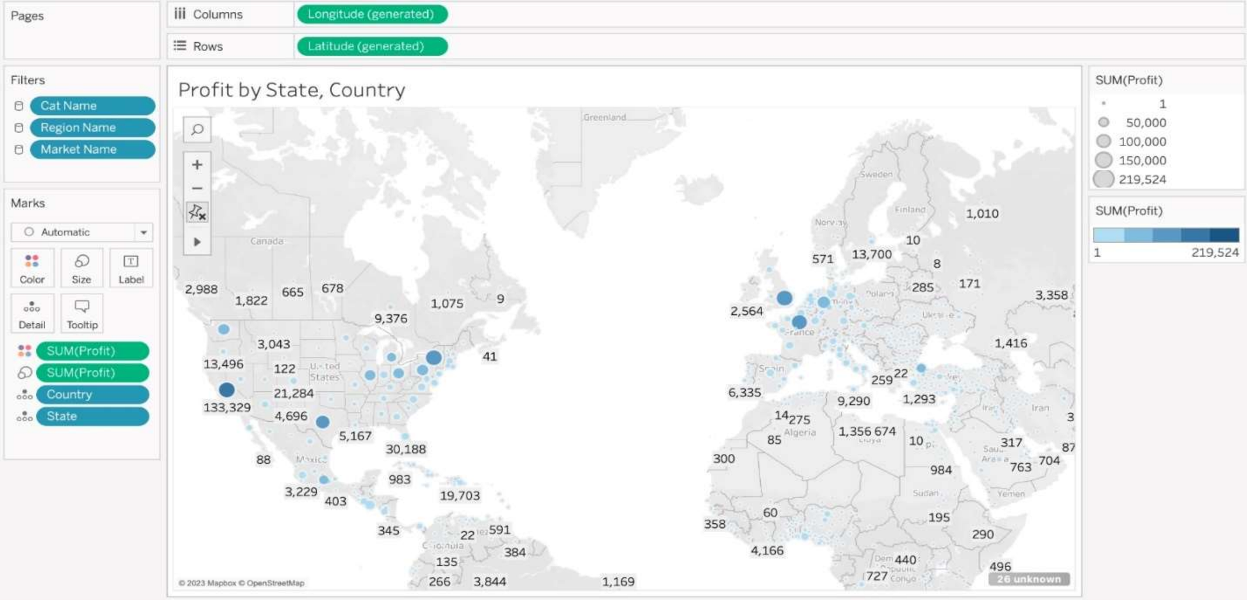
Worksheet 3: Top sales by Sub-Category

Having a graph for this type makes it easier to quickly identify which sub-categories are driving the highest sales figures.



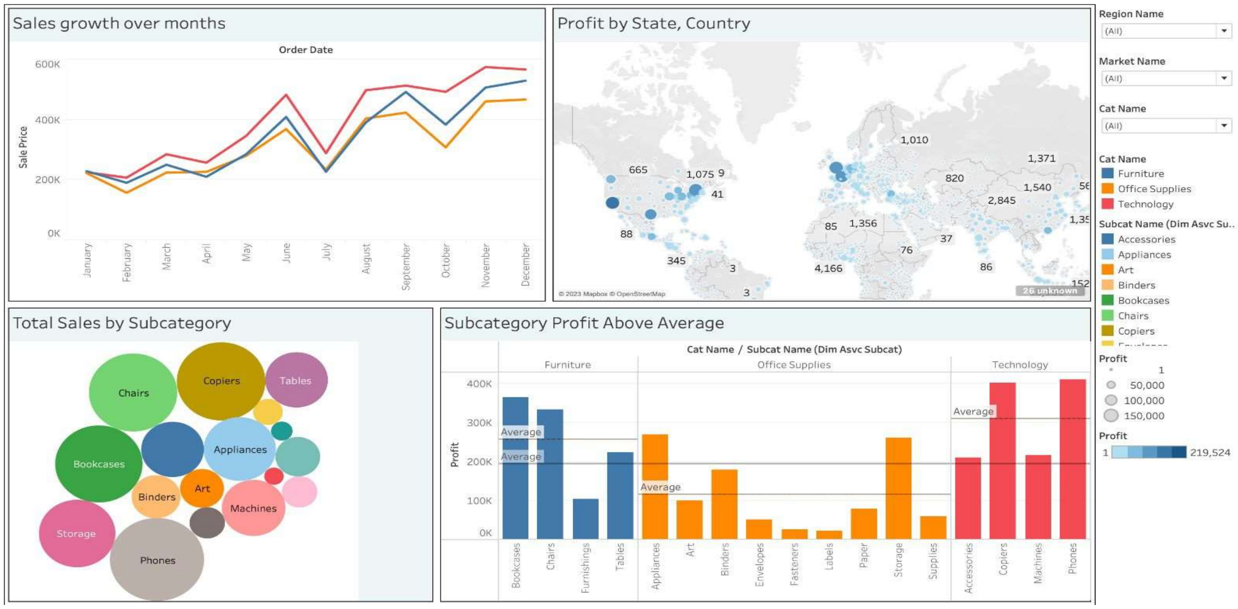
Worksheet 4: Profit by States and Country.

This type of classification by states and country makes it easier to determine and analyze their profit performance in more detail. This can be especially useful for identifying areas of opportunity or concern, as well as for benchmarking performance against other regions or countries.



Interactive dashboard:

We have created an interactive dashboard of multiple worksheets which will allow us to see how sales are changing over time for each region, market, category, etc. The graphs keep changing as we check or uncheck the filters on the right.



Region Name: (All)

Market Name: (All)

Cat Name: (All)

PROJECT INSIGHTS

Throughout the duration of the project, we encountered many challenges and while solving the challenges, we almost always discovered and learned something new. We successfully learned how to use CSV files to load data in the OLTP database. It was challenging to map out the relations correctly and to join the tables in a way that represents the source data. Data Warehousing also presented its share of challenges and learnings, but we were able to learn how and why a DW is implemented. It was exciting to learn to represent data in the form of Fact and Dim tables so that required data could be viewed quickly with just a single join. Incremental ETL was also an intriguing concept as it allowed us to use the resources efficiently and only update the data which was updated or was not present in the DW before, which becomes really important as we scale our database as it is imperative that we do not transfer the same data again periodically. We also learned how to use the AWS S3 bucket to store data and then integrate it with the Snowflake data warehouse. One thing that the project made us realize is that it is not easy to make a flawless end-to-end data pipeline as we often found ourselves unraveling new information at a later stage which required modifications in the previous stages. Finally, after loading the data warehouse with the correct data, we learned how to use that data to provide insights to the end users and decision-makers. We grasped the concepts of Tableau to make relations within the DW tables after linking the platform with Snowflake. This allowed us to derive meaningful visualizations from the data which could be altered with filters according to the needs of the users. Overall, the experience was fruitful as it presented us with many challenges and learnings, and made us better at programming and collaboration in general.

APPENDIX

OLTP DDL:

```
CREATE TABLE asvc_address (     cust_id
VARCHAR2(15) NOT NULL,
market_id   VARCHAR2(4) NOT NULL,
street      VARCHAR2(8) NOT NULL,
city        VARCHAR2(15) NOT NULL,
state       VARCHAR2(15) NOT NULL,
zipcode     VARCHAR2(12) NOT NULL,
country     VARCHAR2(20) NOT NULL,
segment     VARCHAR2(20) NOT NULL,
tbl_last_dt DATE NOT NULL
);

COMMENT ON COLUMN asvc_address.street IS
   'STREET OF THE ADDRESS OF CUSTOMER';

COMMENT ON COLUMN asvc_address.city IS
   'CITY  OF THE ADDRESS OF CUSTOMER';

COMMENT ON COLUMN asvc_address.state IS
   'STATE OF THE ADDRESS OF CUSTOMER';

COMMENT ON COLUMN asvc_address.zipcode IS
   'ZIPCODE OF THE ADDRESS OF CUSTOMER';

COMMENT ON COLUMN asvc_address.country IS
   'COUNTRY OF THE ADDRESS OF CUSTOMER';
```

```
COMMENT ON COLUMN asvc_address.segment IS     'SEGMENT OF THE ADDRESS OF
CUSTOMER';

ALTER TABLE asvc_address ADD CONSTRAINT asvc_address_pk PRIMARY KEY (
market_id,  cust_id );

CREATE TABLE asvc_ctgry (     cat_id
VARCHAR2(4) NOT NULL,     subcat_id
VARCHAR2(4) NOT NULL,     cat_name
VARCHAR2(20) NOT NULL,     descp
VARCHAR2(50),     tbl_last_dt DATE
NOT NULL
);

COMMENT ON COLUMN asvc_ctgry.cat_id IS
   'PRIMARY KEY FOR CATEGORY ENTITY';

COMMENT ON COLUMN asvc_ctgry.cat_name IS
   'NAME OF THE CATEGORY';

COMMENT ON COLUMN asvc_ctgry.descp IS
   'DESCRIPTION OF THE CATEGORY';

ALTER TABLE asvc_ctgry ADD CONSTRAINT asvc_ctgry_pk PRIMARY KEY ( cat_id );

CREATE TABLE asvc_customer (     cust_id
VARCHAR2(15) NOT NULL,     cust_fname
VARCHAR2(10)          NOT          NULL,
cust_mname  VARCHAR2(10),    cust_lname
VARCHAR2(10) NOT NULL,     tbl_last_dt
DATE NOT NULL
);
```

```
COMMENT ON COLUMN asvc_customer.cust_id IS
    'PRIMARY KEY FOR CUSTOMER ENTITY';

COMMENT ON COLUMN asvc_customer.cust_fname IS
    'FIRST NAME OF THE CUSTOMER';

COMMENT ON COLUMN asvc_customer.cust_mname IS
    'CUSTOMER MIDDLE NAME';

COMMENT ON COLUMN asvc_customer.cust_lname IS
    'CUSTOMERS LAST NAME';

ALTER TABLE asvc_customer ADD CONSTRAINT asvc_customer_pk PRIMARY KEY (
cust_id );

CREATE TABLE asvc_market (     market_id
VARCHAR2(4) NOT NULL,     region_id
VARCHAR2(4) NOT NULL,     market_name
VARCHAR2(15) NOT NULL,     tbl_last_dt
DATE NOT NULL
);

COMMENT ON COLUMN asvc_market.market_id IS
    'PRIMARY KEY FOR MARKET ENTITY';

COMMENT ON COLUMN asvc_market.market_name IS
    'MARKET NAME OF CUSTOMER ADDRESS  LOCATED';

ALTER TABLE asvc_market ADD CONSTRAINT asvc_market_pk PRIMARY KEY (
market_id );
```

```
CREATE TABLE asvc_ordr (     ordr_id
VARCHAR2(15) NOT NULL,     cust_id
VARCHAR2(15) NOT NULL,     ordr_date
DATE NOT NULL,     ship_date     DATE
NOT NULL,     ordr_priority VARCHAR2(15)
NOT NULL,     modeof_ship
VARCHAR2(15) NOT NULL,     tbl_last_dt
DATE NOT NULL
);

COMMENT ON COLUMN asvc_ordr.ordr_id IS
   'PRIMARY KEY FOR ORDER ENTITY';

COMMENT ON COLUMN asvc_ordr.ordr_date IS
   'DATE WHEN ORDER WAS PLACED';

COMMENT ON COLUMN asvc_ordr.ship_date IS
   'SHIPPING DATE OR ORDER';

COMMENT ON COLUMN asvc_ordr.ordr_priority IS
   'PRIORITY OF ORDER';

COMMENT ON COLUMN asvc_ordr.modeof_ship IS
   'MODE OF SHIPMENT';

ALTER TABLE asvc_ordr ADD CONSTRAINT asvc_ordr_pk PRIMARY KEY ( ordr_id );

CREATE TABLE asvc_ordr_prdct (
ordr_id     VARCHAR2(15) NOT NULL,
prdct_id    VARCHAR2(15) NOT NULL,
qty         NUMBER(4) NOT NULL,
   ship_cost   NUMBER(4, 2) NOT NULL,
discount    NUMBER(3, 2) NOT NULL,
profit      NUMBER(4, 2) NOT NULL,
sale_price  NUMBER(4, 2) NOT NULL,
tbl_last_dt DATE NOT NULL
);

COMMENT ON COLUMN asvc_ordr_prdct.qty IS
   'QUANTITY OF PRODUCTS';

COMMENT ON COLUMN asvc_ordr_prdct.ship_cost IS
   'SHIPPING COST ';

COMMENT ON COLUMN asvc_ordr_prdct.discount IS
   'DISCOUNT APPLIED ';

COMMENT ON COLUMN asvc_ordr_prdct.profit IS
   'PROFIT EARNED';

COMMENT ON COLUMN asvc_ordr_prdct.sale_price IS
   'SALES PRICE';

ALTER TABLE asvc_ordr_prdct ADD CONSTRAINT asvc_ordr_prdct_pk PRIMARY KEY (
ordr_id, prdct_id );

CREATE TABLE asvc_prdct (     prdct_id
VARCHAR2(15) NOT NULL,     cat_id
VARCHAR2(4) NOT NULL,     prdct_name
```

```
VARCHAR2(15) NOT NULL,
prdct_descp VARCHAR2(50),
   tbl_last_dt DATE NOT NULL
);

COMMENT ON COLUMN asvc_prdct.prdct_id IS
   'PRIMARY KEY FOR PRODUCT
ENTITY';

COMMENT ON COLUMN asvc_prdct.prdct_name IS
   'NAME OF THE PRODUCT';

COMMENT ON COLUMN asvc_prdct.prdct_descp IS
   'DESCRIPTION OF THE PRODUCT';

ALTER TABLE asvc_prdct ADD CONSTRAINT asvc_prdct_pk PRIMARY KEY ( prdct_id );

CREATE TABLE asvc_region (     region_id
VARCHAR2(4) NOT NULL,     region_name
VARCHAR2(15) NOT NULL,     tbl_last_dt
DATE NOT NULL
);

COMMENT ON COLUMN asvc_region.region_id IS
   'PRIMARY KEY FOR REGION ENTITY';
COMMENT ON COLUMN asvc_region.region_name IS
   'REGION NAME WHERE MANY MARKETS ARE LOCATED';

ALTER TABLE asvc_region ADD CONSTRAINT asvc_region_pk PRIMARY KEY (
region_id
);
```

```
CREATE TABLE asvc_return (
   rtrn_id     VARCHAR2(15) NOT NULL,
   ordr_id     VARCHAR2(15) NOT NULL,
returned    CHAR(1) NOT NULL,
tbl_last_dt DATE NOT NULL
);

COMMENT ON COLUMN asvc_return.rtrn_id IS
   'PRIMARY KEY FOR RETURN ENTITY
';
COMMENT ON COLUMN asvc_return.returned IS
   'ITEM IS RETURNED OR NOT';

ALTER TABLE asvc_return ADD CONSTRAINT asvc_return_pk PRIMARY KEY ( rtrn_id );

CREATE TABLE asvc_subcat (     subcat_id
VARCHAR2(4) NOT NULL,     subcat_name
VARCHAR2(15) NOT NULL,     descp
VARCHAR2(20),     tbl_last_dt DATE NOT
NULL
);

COMMENT ON COLUMN asvc_subcat.descp IS
   'DESCRIPTION OF SUBCATEGORY';
ALTER TABLE asvc_subcat ADD CONSTRAINT asvc_subcat_pk PRIMARY KEY (
subcat_id
);

ALTER TABLE asvc_address
   ADD CONSTRAINT asvc_address_asvc_customer_fk FOREIGN KEY ( cust_id )
      REFERENCES asvc_customer ( cust_id );
```

```
ALTER TABLE asvc_address
   ADD CONSTRAINT asvc_address_asvc_market_fk FOREIGN KEY ( market_id )
REFERENCES asvc_market ( market_id );
ALTER TABLE asvc_ctgry
   ADD CONSTRAINT asvc_ctgry_asvc_subcat_fk FOREIGN KEY ( subcat_id )
REFERENCES asvc_subcat ( subcat_id );
ALTER TABLE asvc_market
   ADD CONSTRAINT asvc_market_asvc_region_fk FOREIGN KEY ( region_id )
REFERENCES asvc_region ( region_id );

ALTER TABLE asvc_ordr
   ADD CONSTRAINT asvc_ordr_asvc_customer_fk FOREIGN KEY ( cust_id )
REFERENCES asvc_customer ( cust_id );

ALTER TABLE asvc_ordr_prdct
   ADD CONSTRAINT asvc_ordr_prdct_asvc_ordr_fk FOREIGN KEY ( ordr_id )
REFERENCES asvc_ordr ( ordr_id );

ALTER TABLE asvc_ordr_prdct
   ADD CONSTRAINT asvc_ordr_prdct_asvc_prdct_fk FOREIGN KEY ( prdct_id )
REFERENCES asvc_prdct ( prdct_id );

ALTER TABLE asvc_prdct
   ADD CONSTRAINT asvc_prdct_asvc_ctgry_fk FOREIGN KEY ( cat_id )
      REFERENCES asvc_ctgry ( cat_id );
ALTER TABLE asvc_return
   ADD CONSTRAINT asvc_return_asvc_ordr_fk FOREIGN KEY ( ordr_id )
      REFERENCES asvc_ordr ( ordr_id );
```

OLTP DML:

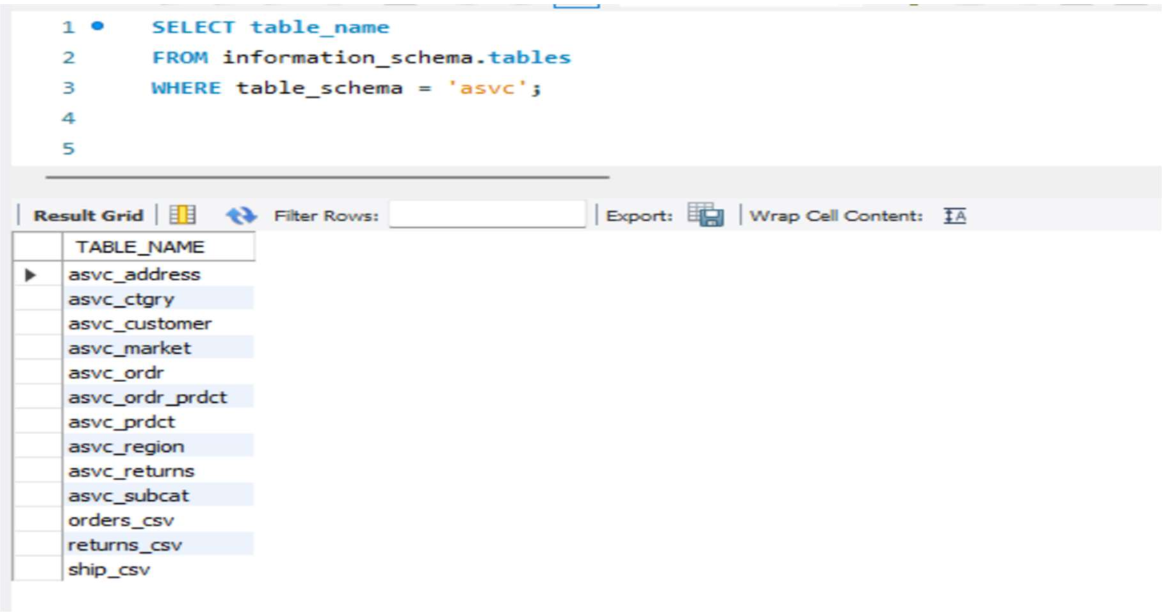
```
INSERT INTO asvc_ctgry (cat_id, cat_name,tbl_last_dt)
SELECT CONCAT('M', LPAD(ROW_NUMBER() OVER(ORDER BY Category), 3, '0')),
Category, NOW()
FROM (SELECT DISTINCT Category FROM orders_csv) AS t;

INSERT INTO asvc_market (market_id, market_name,tbl_last_dt)
SELECT CONCAT('MR', LPAD(ROW_NUMBER () OVER(ORDER BY Market), 3, '0')),
Market, NOW()
FROM (SELECT DISTINCT Market FROM orders_csv) AS t;

INSERT INTO asvc_subcat (subcat_id,cat_id, subcat_name,TBL_LAST_DT)
SELECT CONCAT('S', LPAD(ROW_NUMBER() OVER(ORDER BY subcat), 3, '0')),
m.cat_id, subcat,NOW()
FROM (SELECT DISTINCT subcat, Category FROM orders_Csv) AS t
INNER JOIN asvc_ctgry m ON m.cat_name = t.Category;

INSERT INTO asvc_region (region_id, market_id,region_name,tbl_last_dt)
SELECT CONCAT('R', LPAD(ROW_NUMBER() OVER(ORDER BY Region), 4,
'O')), m.market_id, Region, NOW()
FROM (SELECT DISTINCT Region, Market FROM orders_Csv) AS t
JOIN asvc_market AS m ON t.Market = m.market_name;

INSERT INTO asvc_returns (rtrn_id, ordr_id, returned)
SELECT CONCAT('RE', LPAD(ROW_NUMBER() OVER (), 4, '0')), o.ordr_id, r.returned
FROM returns_csv r
JOIN asvc_ordr o ON r.`Order ID` = o.ordr_id;

INSERT IGNORE INTO asvc_ordr_prdct (ordr_id, mode_of_ship, ship_date, QTY, ship_cost,
Discount, profit, sale_price, PRDCT_ID)
```

```
SELECT o.order_id, o.mode_of_ship, STR_TO_DATE(o.ship_date, '%m/%d/%Y'),
o.QUANTITY, CAST(REPLACE(o.ship_cost, '$', '') AS DECIMAL(10,5)), o.DISCOUNT,
o.Profit, CAST(REPLACE(o.Sales, '$', '') AS DECIMAL(10,5)), p.prdct_id
FROM orders_csv o
INNER JOIN ASVC_ORDR OD ON o.ORDER_ID = OD.ORDR_ID
INNER JOIN asvc_prdct p ON o.Product_ID = p.prdct_id;


INSERT INTO asvc_customer (cust_id, cust_fname, cust_lname, region_id,tbl_last_dt)
SELECT DISTINCT cust_id,
       SUBSTRING_INDEX(cust_name, ' ', 1) as cust_fname,
       SUBSTRING_INDEX(cust_name, ' ', -1) as cust_lname,
       r.region_id, NOW()
FROM orders_csv oc
JOIN asvc_region r ON oc.region = r.region_name;


 INSERT ignore INTO asvc_address (cust_id, city, state, zipcode, country,
segment,tbl_last_dt,region_id)
 SELECT c.cust_id, o.city, o.state, o.zipcode, o.country, o.segment, now(), r.region_id
 FROM (
    SELECT DISTINCT CONCAT(cust_fname, ' ', cust_lname) AS cust_name, cust_id
    FROM asvc_customer
 ) c
 JOIN orders_csv o ON o.cust_id = c.cust_id
 JOIN asvc_region r ON o.region = r.region_name
 WHERE NOT EXISTS (
    SELECT 1 FROM asvc_address a
    WHERE a.cust_id = c.cust_id
      AND a.city = o.city
      AND a.state = o.state
      AND a.zipcode = o.zipcode
      AND a.country = o.country
      AND a.segment = o.segment
```

```
     and a.region_id= r.region_id
 );

INSERT ignore INTO asvc_ordr (ordr_id, cust_id, row_id, ordr_date, ordr_priority, tbl_last_dt)
SELECT DISTINCT o.order_id, c.cust_id, o.row_id,
STR_TO_DATE(o.order_date,'%m/%d/%Y'),
 o.ordr_priority, now()
FROM orders_csv o
JOIN asvc_customer c ON o.cust_id = c.cust_id;

INSERT IGNORE INTO asvc_prdct (prdct_id, prdct_name, cat_id,tbl_last_dt)
SELECT p.p_id, p.prdct_name, c.cat_id,now()
FROM orders_csv p
JOIN asvc_ctgry c ON p.category =c.cat_name;
```

OLTP DATA DICTIONARY QUERY

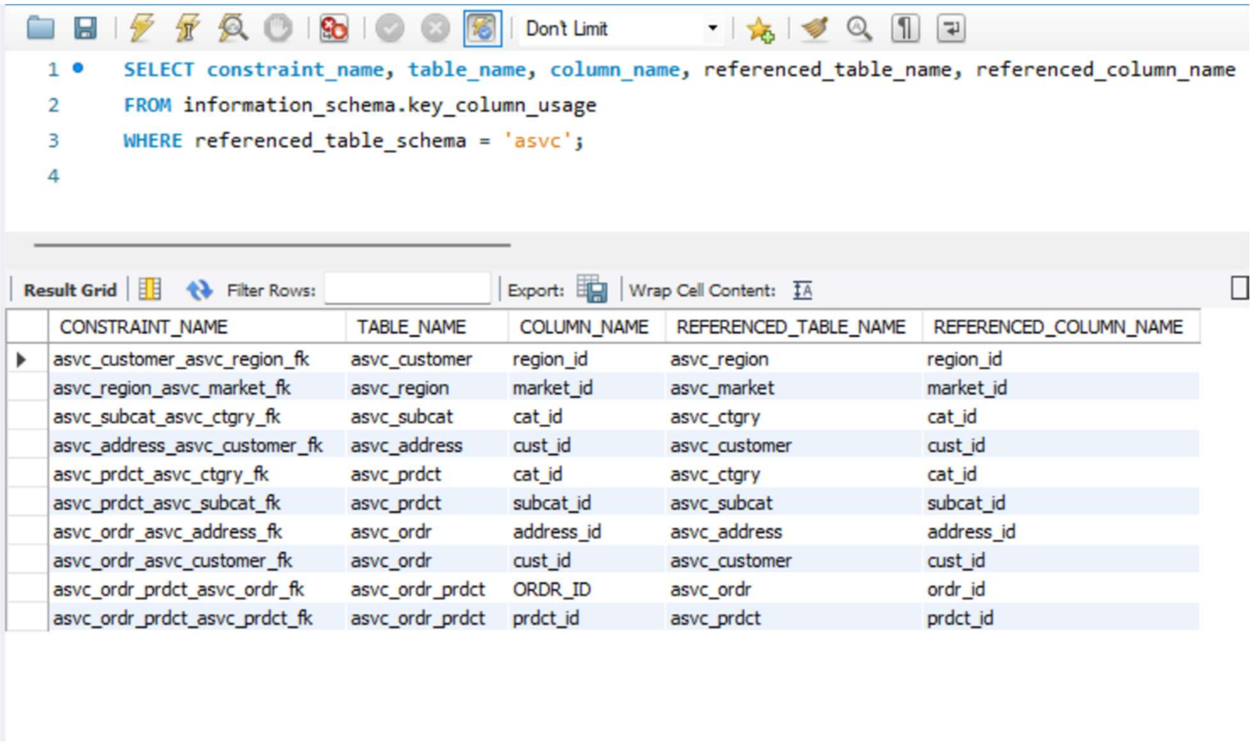
LIST OF TABLES:



```
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'asvc';
```

TABLE_NAME
asvc_address
asvc_ctgry
asvc_customer
asvc_market
asvc_ordr
asvc_ordr_prdct
asvc_prdct
asvc_region
asvc_returns
asvc_subcat
orders_csv
returns_csv
ship_csv

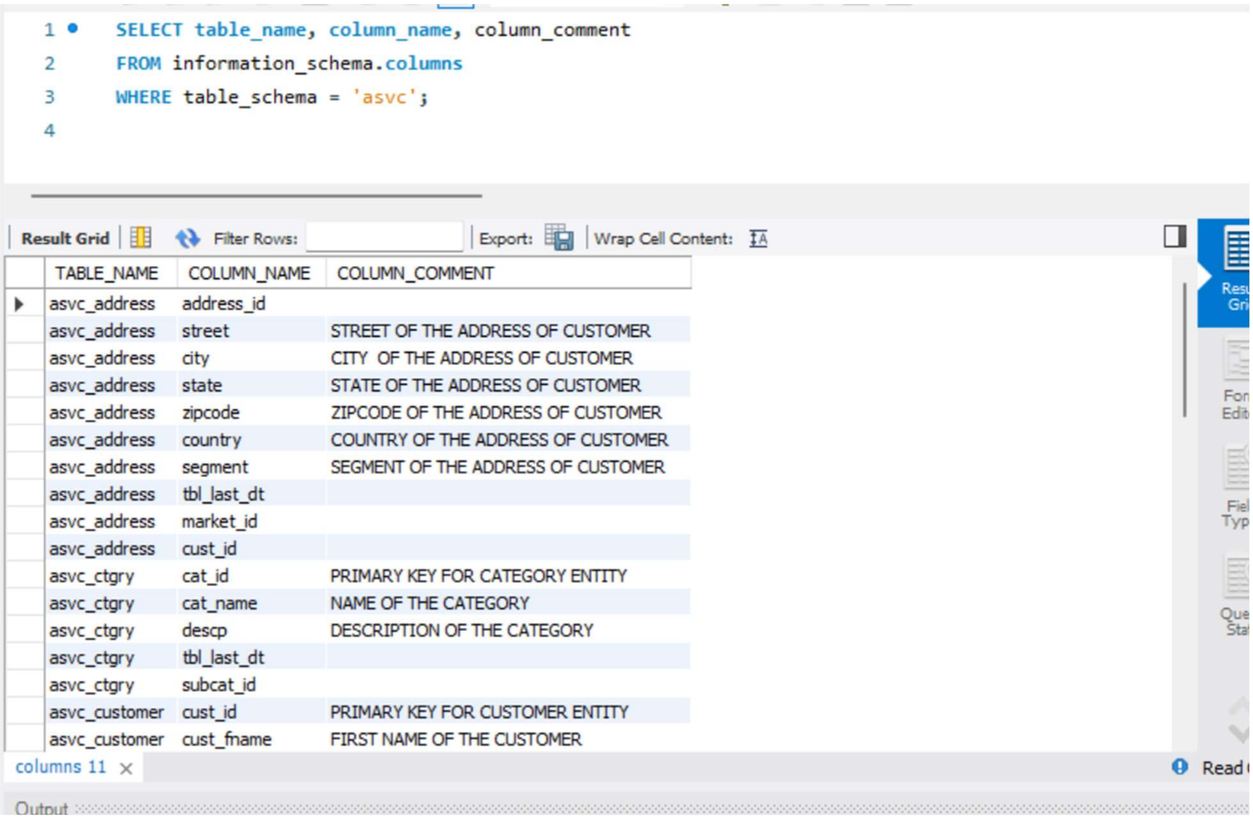
LIST OF CONSTRAINTS FOR ALL TABLES:



```
SELECT constraint_name, table_name, column_name, referenced_table_name, referenced_column_name
FROM information_schema.key_column_usage
WHERE referenced_table_schema = 'asvc';
```

CONSTRAINT_NAME	TABLE_NAME	COLUMN_NAME	REFERENCED_TABLE_NAME	REFERENCED_COLUMN_NAME
asvc_customer_asvc_region_fk	asvc_customer	region_id	asvc_region	region_id
asvc_region_asvc_market_fk	asvc_region	market_id	asvc_market	market_id
asvc_subcat_asvc_ctgry_fk	asvc_subcat	cat_id	asvc_ctgry	cat_id
asvc_address_asvc_customer_fk	asvc_address	cust_id	asvc_customer	cust_id
asvc_prdct_asvc_ctgry_fk	asvc_prdct	cat_id	asvc_ctgry	cat_id
asvc_prdct_asvc_subcat_fk	asvc_prdct	subcat_id	asvc_subcat	subcat_id
asvc_ordr_asvc_address_fk	asvc_ordr	address_id	asvc_address	address_id
asvc_ordr_asvc_customer_fk	asvc_ordr	cust_id	asvc_customer	cust_id
asvc_ordr_prdct_asvc_ordr_fk	asvc_ordr_prdct	ORDR_ID	asvc_ordr	ordr_id
asvc_ordr_prdct_asvc_prdct_fk	asvc_ordr_prdct	prdct_id	asvc_prdct	prdct_id

LIST OF COLUMNS AND COMMENTS FOR EACH TABLE:



```
SELECT table_name, column_name, column_comment
FROM information_schema.columns
WHERE table_schema = 'asvc';
```

TABLE_NAME	COLUMN_NAME	COLUMN_COMMENT
asvc_address	address_id	
asvc_address	street	STREET OF THE ADDRESS OF CUSTOMER
asvc_address	city	CITY OF THE ADDRESS OF CUSTOMER
asvc_address	state	STATE OF THE ADDRESS OF CUSTOMER
asvc_address	zipcode	ZIPCODE OF THE ADDRESS OF CUSTOMER
asvc_address	country	COUNTRY OF THE ADDRESS OF CUSTOMER
asvc_address	segment	SEGMENT OF THE ADDRESS OF CUSTOMER
asvc_address	tbl_last_dt	
asvc_address	market_id	
asvc_address	cust_id	
asvc_ctgry	cat_id	PRIMARY KEY FOR CATEGORY ENTITY
asvc_ctgry	cat_name	NAME OF THE CATEGORY
asvc_ctgry	descp	DESCRIPTION OF THE CATEGORY
asvc_ctgry	tbl_last_dt	
asvc_ctgry	subcat_id	
asvc_customer	cust_id	PRIMARY KEY FOR CUSTOMER ENTITY
asvc_customer	cust_fname	FIRST NAME OF THE CUSTOMER

OLAP DDL:

```
CREATE TABLE dim_asvc_ctgry (
   cat_id   NUMBER(10) NOT NULL,
   cat_name VARCHAR2(20) NOT NULL,
   descp    VARCHAR2(50)
);



ALTER TABLE dim_asvc_ctgry ADD CONSTRAINT dim_asvc_ctgry_pk PRIMARY KEY (
cat_id );

CREATE TABLE dim_asvc_customer (
   cust_id    NUMBER(10) NOT NULL,
   address_id NUMBER(15) NOT NULL,
   cust_fname VARCHAR2(200) NOT NULL,
   cust_mname VARCHAR2(100),
   cust_lname VARCHAR2(200) NOT NULL,
   street     VARCHAR2(200) NOT NULL,
   city       VARCHAR2(100) NOT NULL,
   state      VARCHAR2(100) NOT NULL,
   zipcode    VARCHAR2(10) NOT NULL,
   country    VARCHAR2(100) NOT NULL,
   segment    VARCHAR2(50) NOT NULL
);

ALTER TABLE dim_asvc_customer ADD CONSTRAINT dim_asvc_customer_pk PRIMARY
KEY ( address_id,
                                                             cust_id );

CREATE TABLE dim_asvc_date (
   date_id          NUMBER(10) NOT NULL,
   entry_id          NUMBER(10),
   full_date         DATE,
   days              NUMBER(5),
   month_short       VARCHAR2(5),
   month_num         NUMBER(3),
   month_long        VARCHAR2(10),
   day_of_week_short VARCHAR2(5),
   day_of_week_long  VARCHAR2(10),
   year              VARCHAR2(5),
   quarter           VARCHAR2(10)
);


ALTER TABLE dim_asvc_date ADD CONSTRAINT dim_asvc_date_pk PRIMARY KEY (
date_id );

CREATE TABLE dim_asvc_market (
   market_id   NUMBER(10) NOT NULL,
   market_name VARCHAR2(15) NOT NULL
);


ALTER TABLE dim_asvc_market ADD CONSTRAINT dim_asvc_market_pk PRIMARY
KEY ( market_id );

CREATE TABLE dim_asvc_prdct (
   prdct_id    NUMBER(10) NOT NULL,
   prdct_name  VARCHAR2(15) NOT NULL,
   prdct_descp VARCHAR2(50)
);
```

```
ALTER TABLE dim_asvc_prdct ADD CONSTRAINT dim_asvc_prdct_pk PRIMARY KEY (
prdct_id );

CREATE TABLE dim_asvc_region (
    region_id   NUMBER(10) NOT NULL,
    region_name VARCHAR2(20) NOT NULL,
    tbl_last_dt DATE NOT NULL,
    market_id   VARCHAR2(4) NOT NULL
);



ALTER TABLE dim_asvc_region ADD CONSTRAINT dim_asvc_region_pk PRIMARY KEY
( region_id );

CREATE TABLE dim_asvc_subcat (
    subcat_id   NUMBER(10) NOT NULL,
    subcat_name VARCHAR2(15) NOT NULL,
    descp       VARCHAR2(20)
);

ALTER TABLE dim_asvc_subcat ADD CONSTRAINT dim_asvc_subcat_pk PRIMARY KEY
( subcat_id );

CREATE TABLE fact_asvc_order (
    "ROWNUM"       NUMBER(15) NOT NULL,
    order_date     DATE NOT NULL,
    order_priority VARCHAR2(20) NOT NULL,
    qty            NUMBER(10) NOT NULL,
    ship_cost      NUMBER(15) NOT NULL,
    ship_date      DATE NOT NULL,
    discount       NUMBER(3, 2) NOT NULL,
    mode_of_ship   VARCHAR2(20) NOT NULL,
    profit        NUMBER(4, 2) NOT NULL,
    sale_price    NUMBER(4, 2) NOT NULL,
    returned      CHAR(1) NOT NULL,
    date_id       NUMBER(10) NOT NULL,
    market_id     NUMBER(10) NOT NULL,
    region_id     NUMBER(10) NOT NULL,
    address_id    NUMBER(15) NOT NULL,
    subcat_id     NUMBER(10) NOT NULL,
    prdct_id      NUMBER(10) NOT NULL,
    cat_id        NUMBER(10) NOT NULL,
    cust_id       NUMBER(10) NOT NULL
);

ALTER TABLE fact_asvc_order ADD CONSTRAINT fact_asvc_order_pk PRIMARY KEY (
"ROWNUM" );

ALTER TABLE fact_asvc_order
    ADD CONSTRAINT fact_order_dim_ctgry_fk FOREIGN KEY ( cat_id )
        REFERENCES dim_asvc_ctgry ( cat_id );

ALTER TABLE fact_asvc_order
    ADD CONSTRAINT fact_order_dim_customer_fk FOREIGN KEY ( address_id,
                                                            cust_id )
        REFERENCES dim_asvc_customer ( address_id,
                                       cust_id );

ALTER TABLE fact_asvc_order
    ADD CONSTRAINT fact_order_dim_date_fk FOREIGN KEY ( date_id )
        REFERENCES dim_asvc_date ( date_id );

ALTER TABLE fact_asvc_order
    ADD CONSTRAINT fact_order_dim_market_fk FOREIGN KEY ( market_id )
    REFERENCES dim_asvc_market ( market_id );

ALTER TABLE fact_asvc_order
  ADD CONSTRAINT fact_order_dim_prdct_fk FOREIGN KEY ( prdct_id )
    REFERENCES dim_asvc_prdct ( prdct_id );

ALTER TABLE fact_asvc_order
  ADD CONSTRAINT fact_order_dim_region_fk FOREIGN KEY ( region_id )
    REFERENCES dim_asvc_region ( region_id );

ALTER TABLE fact_asvc_order
  ADD CONSTRAINT fact_order_dim_subcat_fk FOREIGN KEY ( subcat_id )
    REFERENCES dim_asvc_subcat ( subcat_id );
```

PROCEDURE CODE FOR INCREMENTAL ETL:

```
CREATE OR REPLACE PROCEDURE load_merge_fact_asvc_order()
RETURNS VARCHAR
LANGUAGE JAVASCRIPT
AS $$
  var err_code;
  var err_msg;
  try {
    var stmt = snowflake.createStatement({sqlText:`
      MERGE INTO fact_asvc_order a
      USING stg_incrmntl_fact_table b
      ON (a.ordr_id = b.ordr_id) AND (a.prdct_id = b.prdct_id)
      WHEN MATCHED THEN
        UPDATE SET
          a.order_date = b.order_date,
          a.order_priority = b.order_priority,
          a.qty = b.qty,
          a.ship_date = b.ship_date,
          a.ship_cost = b.ship_cost,
          a.mode_of_ship = b.mode_of_ship,
          a.discount = b.discount,
          a.profit = b.profit,
          a.sale_price = b.sale_price,
          a.address_id = b.address_id,
          a.market_id = b.market_id,
          a.region_id = b.region_id,
          a.cat_id = b.cat_id,
          a.cust_id = b.cust_id,
          a.subcat_id = b.subcat_id,
          a.subcat_name = b.subcat_name,
        a.returned = b.returned
      WHEN NOT MATCHED THEN
        INSERT (ordr_id, order_date, order_priority, qty, ship_date, ship_cost, mode_of_ship, discount, profit,
        sale_price, address_id, market_id, region_id, prdct_id, cat_id, cust_id, subcat_id, subcat_name, returned)
        VALUES (b.ordr_id, b.order_date, b.order_priority, b.qty, b.ship_date, b.ship_cost, b.mode_of_ship, b.discount,
        b.profit, b.sale_price, b.address_id, b.market_id, b.region_id, b.prdct_id, b.cat_id, b.cust_id, b.subcat_id, b.subcat_name, b.returned);
    `});
    var result = stmt.execute();
    return result.next() ? result.getColumnValue(1).toString() : 'Success';
  } catch (err) {
    err_code = err.code;
    err_msg = err.message;
    return 'Error code ' + err_code + ': ' + err_msg;
  }
$$;

CALL load_merge_fact_asvc_order();
```

HISTORY TABLE TRIGGER

```
ALTER TABLE asvc_address_history ADD CONSTRAINT pk_asvc_address_history
PRIMARY KEY (address_id);
DELIMITER $$
CREATE TRIGGER tr_asvc_address_history BEFORE DELETE ON asvc_address
FOR EACH ROW
BEGIN
  INSERT INTO asvc_address_history SELECT * FROM asvc_address WHERE
address_id=OLD.address_id;
  UPDATE asvc_address_history
  SET tbl_last_dt=NOW()
  WHERE address_id=OLD.address_id;
END $$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER tr_asvc_address_history3 after delete ON asvc_address
FOR EACH ROW
BEGIN
  INSERT INTO asvc_address_history SELECT * FROM asvc_address WHERE
address_id=OLD.address_id;
  UPDATE asvc_address_history
  SET tbl_last_dt=NOW()
  WHERE address_id=OLD.address_id;
END $$
DELIMITER ;
```

TABLE PARTITIONING(CLUSTERING IN SNOWFLAKE)

```
CREATE TABLE partitioning_fact_asvc cluster by (MARKET_ID) AS
(SELECT * FROM fact_asvc_order);

SELECT SYSTEM$CLUSTERING_INFORMATION('partitioning_fact_asvc');

SHOW TABLES LIKE 'partitioning_fact_asvc';
```

ETL CODE for dim_asvc_ctgry:

```
create schema asvc_project

CREATE OR REPLACE STORAGE integration aws_s3_integration
type = external_stage
storage_provider='S3'
enabled=true
storage_aws_role_arn= 'arn:aws:iam::649069303992:role/asvc-project'
storage_allowed_locations= ('s3://asvc-project/');

GRANT USAGE ON INTEGRATION aws_s3_integration TO ROLE accountadmin;

create or replace file format demo_format
type='CSV'
field_delimiter='|'
skip_header=1;



CREATE OR REPLACE STAGE demo_aws_stage
storage_integration = aws_s3_integration
file_format = demo_format
url='s3://asvc-project/';

CREATE TABLE dim_asvc_ctgry (
   cat_id   VARCHAR2(10) NOT NULL,
   cat_name VARCHAR2(20) NOT NULL,
   descp    VARCHAR2(50)
);
```

```
COPY INTO dim_asvc_ctgry
FROM @demo_aws_stage/PROJECT/asvc_ctgry.csv
FILE_FORMAT=(format_name=demo_format1 error_on_column_count_mismatch=false)
ON_ERROR = 'CONTINUE';
```