1. Create a Data Model for the Data warehouse.

Based on the the requirements, the following data model can be designed for the data warehouse:

A screenshot of a computer screen

Description automatically generated

**Fact Tables:**

1. **Inventory**

* inventory\_id
* inventory\_product\_id
* inventory\_quantity
* inventory\_location

1. **order**

* order\_id
* order\_customer\_id
* order\_total\_price
* order\_date
* order\_status
* order\_shipping\_address
* order\_billing\_address
* order\_delivery\_date
* order\_payment\_method
* order\_payment\_status
* order\_payment\_date
* order\_shipping\_method
* order\_shipping\_status
* order\_shipping\_date

1. **order\_item**

* order\_id
* order\_product\_id
* order\_quantity

**Dimension Tables:**

1. **Customer**

* customer\_id
* customer\_name
* customer\_email
* customer\_phone

1. **product**

* product\_id
* product\_name
* product\_customer\_price
* product\_purchasing\_price

**Relationships:**

**Customer to order:** One customer can place many orders**.** (1 to Many)

This is represented by **order\_customer\_id** in the **order** table referencing **customer\_id** in the Customer table.

**order to order\_item**: One order can have many order items.(1 to Many)

This is represented by **order\_id** in the **order\_item** table referencing **order\_id** in the **order** table.

**product to order\_item**: One product can be part of many order items. (1 to Many)

This is represented by **order\_product\_id** in the **order\_item** table referencing **product\_id** in the **product** table.

**product to Inventory**: One product can be stored in many inventory locations.(1 to Many)

This is represented by **inventory\_product\_id** in the Inventory table referencing **product\_id** in the **product** table.

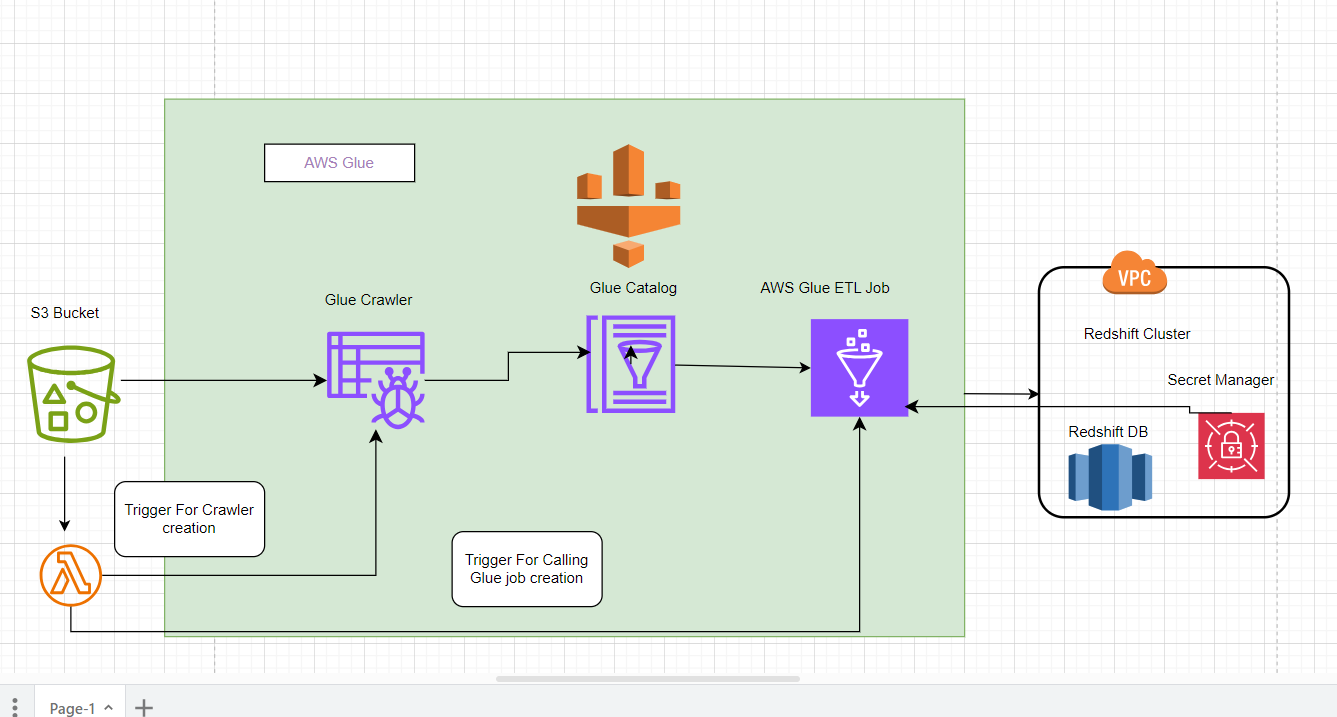
**Explanation:**

* **order fact table:** This table captures the details of each order, including the customer, products, and order-related information. It is a fact table because it represents a measurable event (an order) that occurred at a specific point in time.
* **Inventory fact table:** This table captures the details of the products in different inventory locations. It is a fact table because it represents a measurable state (the quantity of products in each location) that can be used to analyze inventory levels and manage stock.
* **order\_item fact table:** This table captures the details of the products ordered in each order. It is a fact table because it represents a measurable event (the order of products) that occurred at a specific point in time.
* **Customer dimension table:** This table captures the details of each customer, including their name, email, and phone number. It is a dimension table because it provides context about the entities involved in the orders.
* **product dimension table:** This table captures the details of each product, including its name, customer price, and purchasing price. It is a dimension table because it provides context about the products involved in the orders and inventory.

**Summary:**

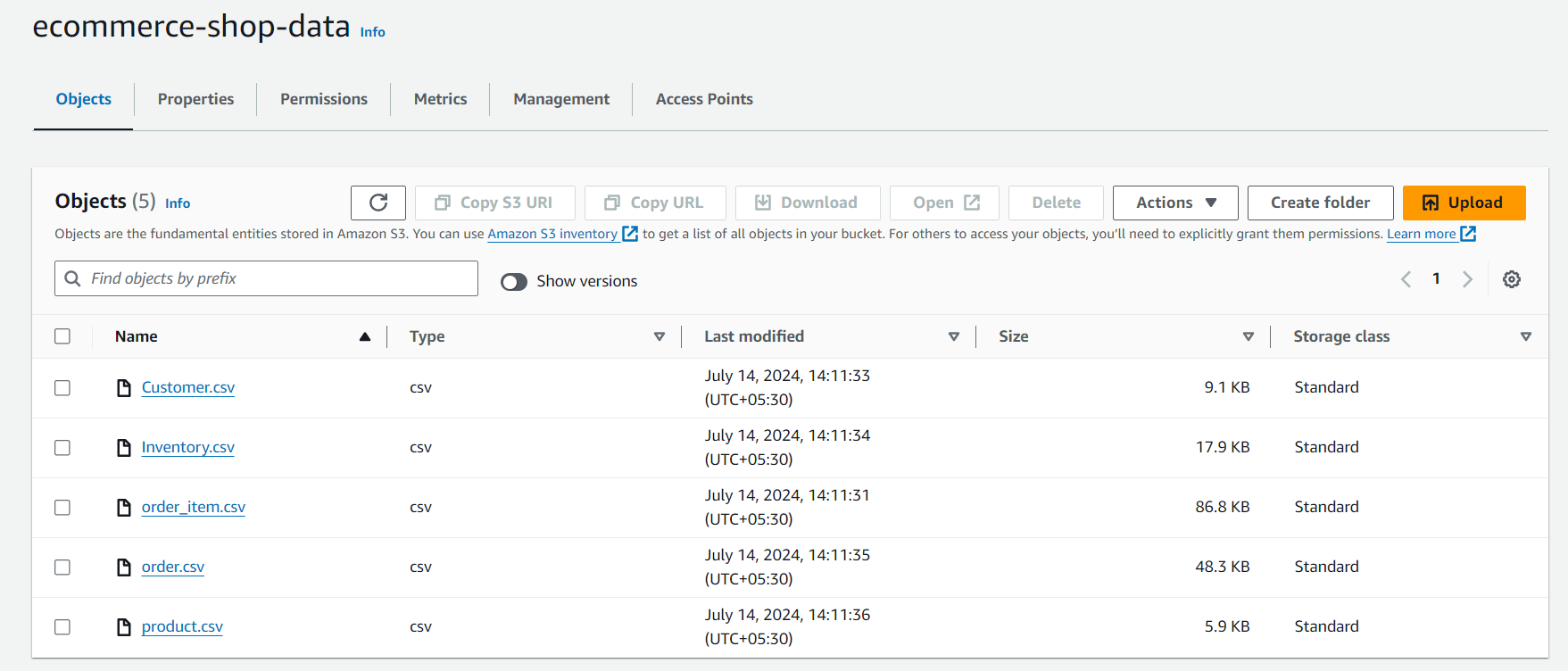
The **order, Inventory** and **order\_item** tables are fact tables because they capture measurable events or states that can be used to analyze ecommerce shop’s operations. The **customer** and **product** tables are dimension tables because they provide context about entities involved in the order and Inventory.

2.Create a data pipeline to load the data into the data warehouse.



As per the above Diagram I have used the following service to achieve this requirement.

1. **S3 Bucket**: The CSV files are stored in S3 bucket.



1. **Lambda Function**: When a new CSV file has been uploaded, the lambda function will be triggered for creating Glue Crawler.

ecommerce-datawarehouse/lambda/ data\_pipeline\_trigger.py

A screenshot of a computer

Description automatically generated

1. **Glue Crawler**: The Lambda function creates a new Glue crawler, which is configured to crawl the S3 bucket and the CSV file(s) that triggered the Lambda function. So, key advantages of using AWS Glue Crawler are that they can automatically create databases and tables in the AWS Glue Data Catalog without manual intervention.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

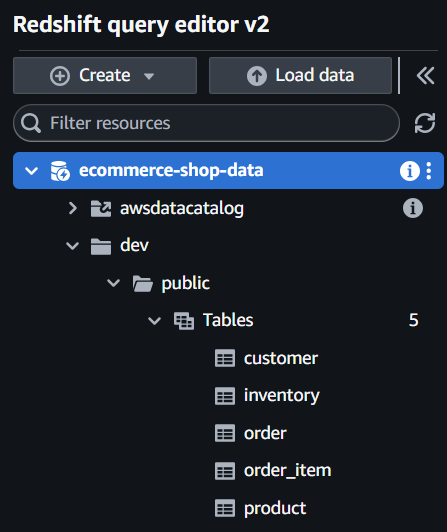
Description automatically generated

1. **Glue Job**: After the crawler is created and has processed the CSV file(s), the Lambda function triggers a Glue job which will reads the data from the source (**S3 bucket**) based on the metadata in the Glue Data Catalog, performs necessary transformations, and loads the data into the target data store (**Amazon Redshift**). For connecting AWS Glue and Redshift cluster we need username and password so for making more secure manner we have used secret manager and passed an argument in Glue job.

**Glue Job** has been written at the following path in project directory

**ecommerce-datawarehouse/glue/etl.py**

1. **Redshift:** Once the data is loaded into Redshift then we can do further analysis and querying on the data using SQL queries.



1. Create a SQL query to calculate the revenue per customer.

Required query has been written and it can be found under the following project directory

**ecommerce-datawarehouse/redshift/ revenue\_per\_customer.sql**

1. Create a SQL query to calculate the value of goods stored in the different inventory locations.

Required query has been written and it can be found under the following project directory

**ecommerce-datawarehouse/redshift/** **inventory\_value.sql**

1. Create a SQL query to calculate the average order value per customer.

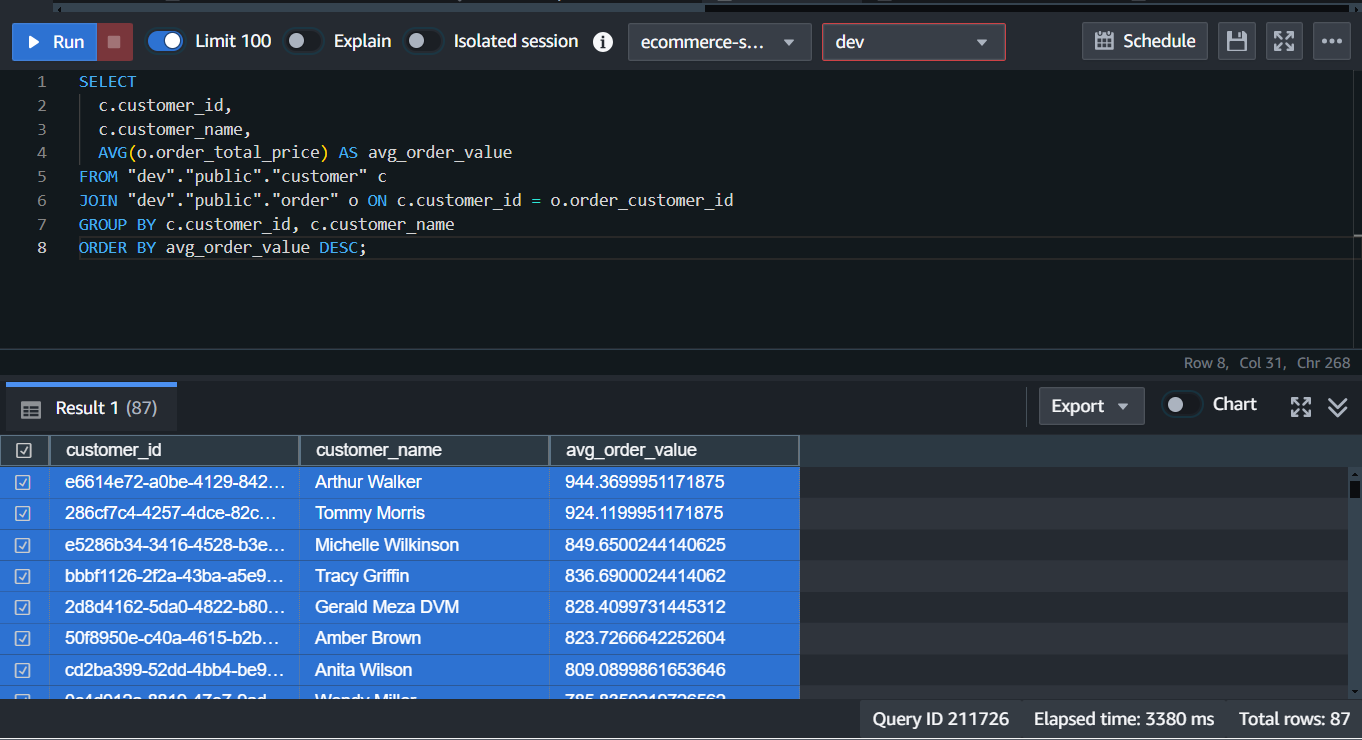
Required query has been written and it can be found under the following project directory

**ecommerce-datawarehouse/redshift/** **avg\_order\_value.sql**

1. Run the data pipeline and execute the SQL queries on the datawarehouse.

Regarding this thing as mentioned in Question 2 pipeline has been created and data has been loaded into datawarehouse Redshift and perform all query as mentioned in Question 3 to Question 5.

For one of the query please refer below screenshot.



1. Have some thoughts on how to improve the usage of the data warehouse.
2. **Optimize Data Model and Schema:** Review the data model and make necessary changes to improve performance, organization and analysis.
3. **Implement partitioning and clustering:** Partition fact tables and cluster tables to optimize query execution.
4. **Monitor and optimize performance:** Set up monitoring and implement optimizations to improve data warehouse performance.
5. **Continuously Evolve the Datawarehouse:** Regularly review usage patterns and business needs to identify opportunities for further optimization and enhancement.
6. **Ensure Data Governance and Security**: Establish data policies, access controls, and data masking to protect sensitive information.
7. **Enhance Exploration and Visualization**: Integrate with BI tools and develop custom dashboards to enable interactive data analysis.