ECE-GY-9953: Advanced Project I, Fall 2023

Project Part 1 Submission

Team Members:
Brian Mohammed Catraguna,
Palak Pramod Keni

Submitted by:
Palak Pramod Keni
Net ID: pk2539

A. Project Background

This database-driven web application project focuses on building an end-to-end system for a business case study related to sales and returns data, consisting of an online transaction processing (OLTP) database, web application, ETL workflows and a data warehouse optimized for business analytics.

The project required parsing complex CSV files with raw transactional data provided in the business case study and processing the data for the OLTP database schema. The OLTP database supports core business operations via an web application for order processing, inventory control and customer management. It enforces data integrity with normalized schema, checks, triggers, stored procedures and indexes. The web application provides an intuitive user interface and streamlined workflows for customers. The data warehouse was assembled with the Extract Transform Load (ETL) process using Change Data Capture (CDC) method which enables historical analysis, visualizations and reports on key metrics.

Deliverables include well-documented logical and physical data models along with schema definitions, entity relations diagrams, flowcharts documenting architectures and key sequences. All critical stages have test cases and expected outputs enumerated. Learnings in securing sensitive business data while supporting analytics has value for future enterprise initiatives as data volumes explode.

B. Business Case

The business case study involves transactional datasets for orders and returns of products by customers of Awesome Superstore Inc. The business mainly sells Furniture, Office Supplies and Technology products and their customers are the mass Consumer, Corporate and Home Offices. The first dataset provides records of order details which includes customer, product, and order information. The second dataset contains records of returned orders.

Awesome Superstore Inc. has requested the development of an online transaction processing (OLTP) database and web application to facilitate key business functions. The OLTP database will enable real-time processing of transactions including customer onboarding, product data management, and product purchases. The web application will provide the user interface for customers to create accounts, browse products, and complete orders online. This system aims to improve business operations by consolidating multiple data sources, enabling self-service for customers. The development of the OLTP database and web application will digitalize and optimize core retail processes for Awesome Superstore Inc.

Additionally, there is a requirement for developing a data warehouse solution to enable advanced analytics and data-driven decision making. A data warehouse will consolidate transaction data from the OLTP databases and other enterprise data sources into a centralized repository optimized for analytical querying. This data consolidation will empower business users to uncover trends, exceptions and insights from current and historical data to guide future strategic decisions regarding marketing campaigns, store locations, manufacturing capacity, product portfolio, and overall financial performance.

C. Project Milestones

1. Designing the Database Schema (10/07/2023):

On October 7, 2023, the project commenced with the fundamental step of designing the Database Schema. This phase involved creating a logical, relational database structure and defining data using Data Definition Language (DDL). Furthermore, it included the crucial task of populating the database tables with data sourced from CSV files. Establishing a robust database foundation is essential for the smooth functioning of the web application.

2. Database Schema Refinement and Backend-Frontend Setup (10/21/2023):

By October 21, 2023, the project had evolved to address the refinement of the initial database schema. Additionally, efforts were made to configure the Backend and Frontend code to support a REST API. This phase also marked the beginning of the design planning process, where decisions were made regarding which screens and features would be implemented. This planning stage served as a blueprint for the upcoming development work.

3. Feature Building (11/04/2023):

The next milestone, reached on November 4, 2023, signified the start of actual feature development. With the database and codebase in place, the project team embarked on building the core functionality of the web application. Features such as user interfaces, data processing, and application logic began to take shape, marking significant progress towards project completion.

4. Data Warehouse Design and Development (11/18/2023):

As the web application features started to come together, the project entered the phase of Data Warehouse design. This parallel effort aimed to enhance data storage and analytics capabilities while the web application development continued. Developing a robust Data Warehouse was crucial for handling large volumes of data effectively.

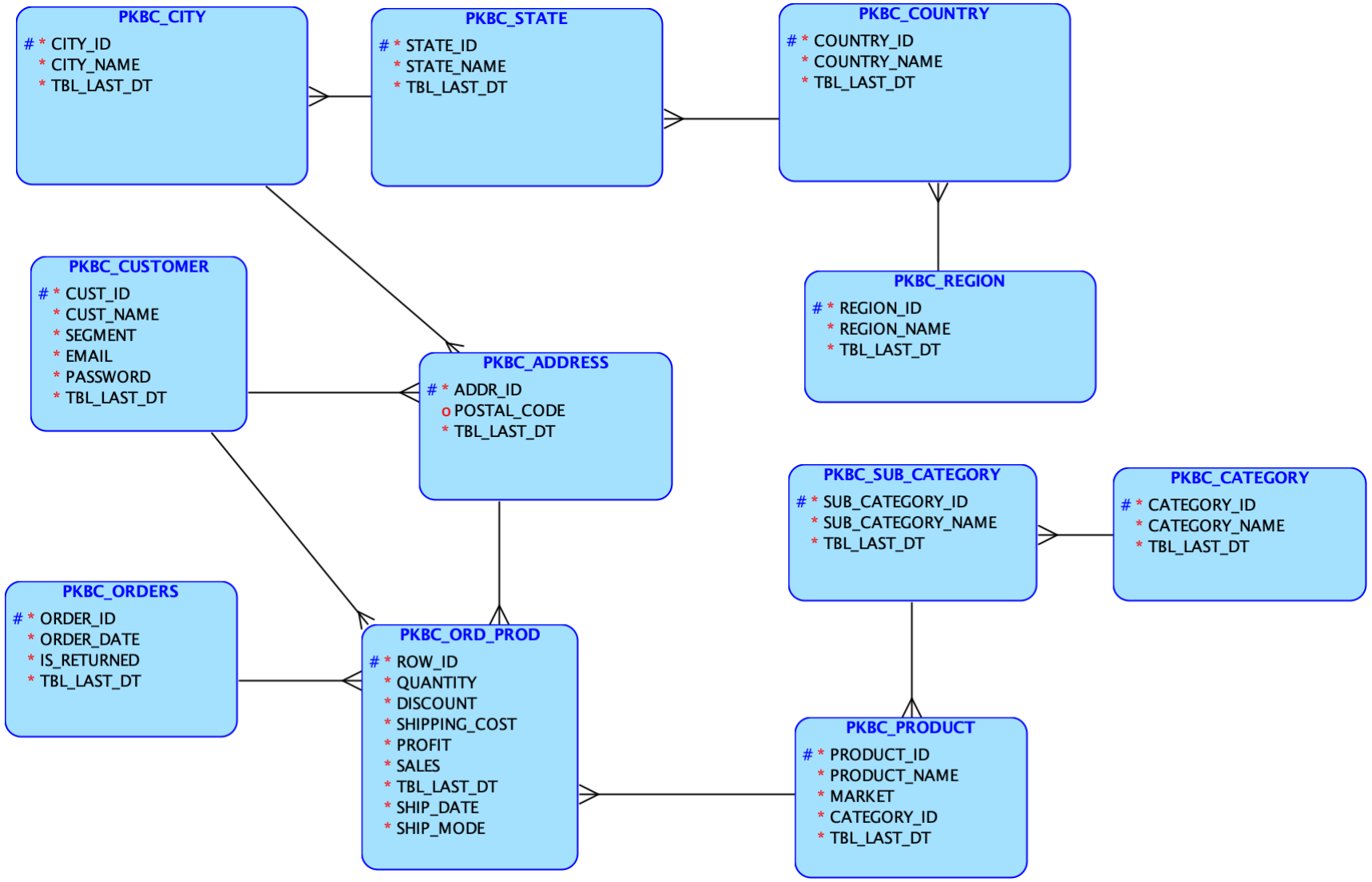
5. Data Warehouse and ETL Implementation (12/02/2023):

December 2, 2023, marked a crucial juncture as the Data Warehouse and Extract, Transform, Load (ETL) scripts were implemented. This step was undertaken with the goal of ensuring that the Data Warehouse was fully operational and could handle the data requirements of the web application. This integration played a pivotal role in achieving a comprehensive and data-driven web application.

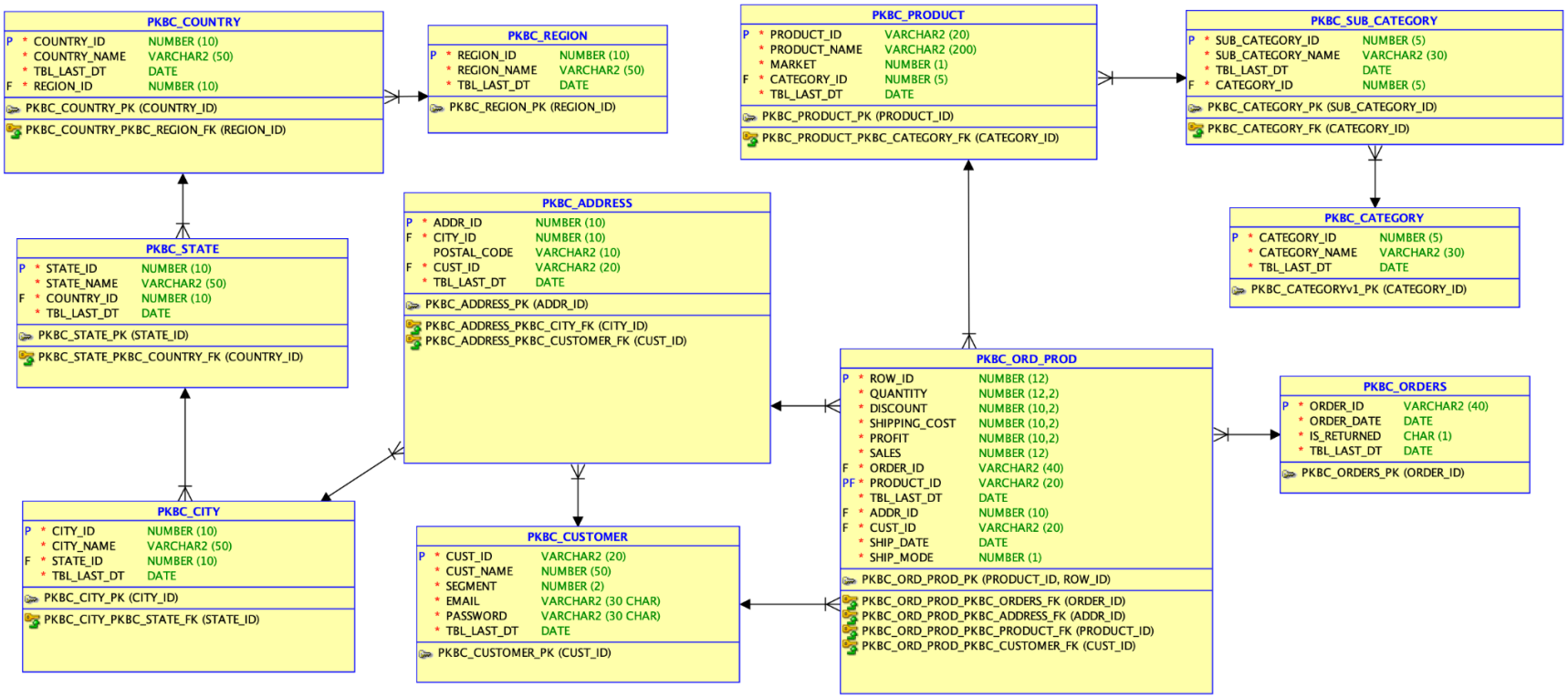
6. Tableau Analysis and Finalization (12/16/2023):

The final milestone, reached on December 16, 2023, included the completion of Tableau analysis. This step allowed for a thorough examination of data insights and reporting capabilities. Additionally, it marked the finalization of the Online Transaction Processing (OLTP) Database, web application, and Data Warehouse. This comprehensive assessment ensured that all components were ready for deployment and actual use.

D. Logical Model(OLTP)


PKBC_CITY
* CITY_ID
* CITY_NAME
* TBL_LAST_DT
PKBC_STATE
* STATE_ID
* STATE_NAME
* TBL_LAST_DT
PKBC_COUNTRY
* COUNTRY_ID
* COUNTRY_NAME
* TBL_LAST_DT
PKBC_CUSTOMER
* CUST_ID
* CUST_NAME
* SEGMENT
* EMAIL
* PASSWORD
* TBL_LAST_DT
PKBC_ADDRESS
* ADDR_ID
o POSTAL_CODE
* TBL_LAST_DT
PKBC_REGION
* REGION_ID
* REGION_NAME
* TBL_LAST_DT
PKBC_SUB_CATEGORY
* SUB_CATEGORY_ID
* SUB_CATEGORY_NAME
* TBL_LAST_DT
PKBC_CATEGORY
* CATEGORY_ID
* CATEGORY_NAME
* TBL_LAST_DT
PKBC_ORDERS
* ORDER_ID
* ORDER_DATE
* IS_RETURNED
* TBL_LAST_DT
PKBC_ORD_PROD
* ROW_ID
* QUANTITY
* DISCOUNT
* SHIPPING_COST
* PROFIT
* SALES
* TBL_LAST_DT
* SHIP_DATE
* SHIP_MODE
PKBC_PRODUCT
* PRODUCT_ID
* PRODUCT_NAME
* MARKET
* CATEGORY_ID
* TBL_LAST_DT

E. Relational Model(OLTP)



F. Assumptions

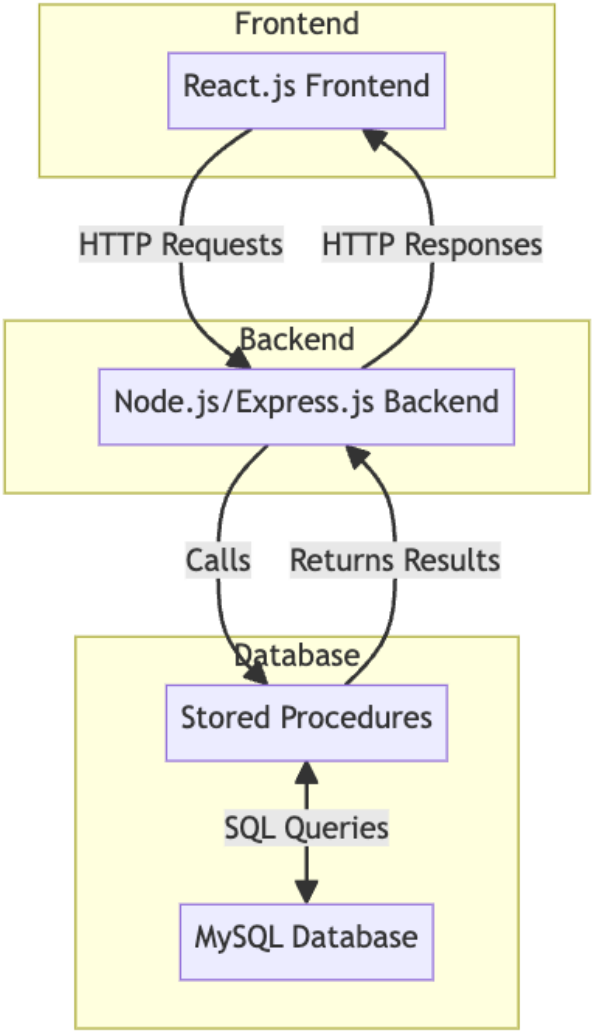
- Each customer can have multiple addresses
- Customer can order multiple products and products can be ordered by multiple customers
- One address is used per order

- One address belongs to one city, one city belongs to many addresses
- One state can have many cities
- One country can have many states
- One region can have many countries
- One subcategory can belong to many products
- One category can belong to many subcategories

G. Constraint

- All products within an order are returned all at once.

H. Infrastructure



The architecture presented in the provided diagram represents the technological stack employed by our project, Awesome Superstore Inc. This architectural framework consists of different layers and technologies, each serving a unique purpose to ensure a seamless and efficient online business operation.

Database Layer: MySQL

At the core of this architecture lies the database layer, implemented using MySQL. MySQL is a well-established relational database management system that offers robust support for data storage and retrieval. For Awesome Superstore Inc., which manages a vast amount of transactional data involving sellers, customers, products, and orders, MySQL is an excellent choice. These are the reasons why:

- **Stored Procedures**: MySQL allows the implementation of stored procedures, which are pre-defined SQL scripts that can be executed by the database. Awesome Superstore Inc. takes advantage of this feature to encapsulate complex business logic within the database itself. These stored procedures not only enhance security by preventing SQL injection but also improve maintainability.
- **Database Indexing**: To optimize query performance, Awesome Superstore Inc. wisely employs database indexing on frequently queried columns. By indexing key columns, the system can rapidly locate and retrieve relevant data. This is particularly critical for the speedy retrieval of product information, order details, and customer data.
- **Transaction Management**: MySQL supports transaction management, enabling the implementation of ACID properties (Atomicity, Consistency, Isolation, Durability). Transactions are crucial for maintaining data integrity when dealing with the numerous financial transactions and updates occurring within the system.
- **Triggers**: The use of triggers to insert and update 'tbl_last_dt' on every table is a strategic choice. These triggers help record the last time a row was updated, essential for Extract, Transform, Load (ETL) processes, and data auditing.

Backend Server Layer: Node.js & Express.js

The backend server, powered by Node.js and Express.js, is responsible for handling HTTP requests and acts as the bridge for interaction between the frontend client and the database. Node.js is a server-side runtime environment, while Express.js is a lightweight and flexible web application framework for Node.js. The selection of this tech stack is commendable for several reasons:

- **Concurrency and Scalability**: Node.js excels at handling a large number of concurrent connections, making it ideal for an e-commerce platform like Awesome Superstore Inc. that expects a high volume of user interactions. It is also well-suited for applications that require real-time communication and responsiveness.

- **REST API Development**: Express.js simplifies the creation of RESTful APIs. REST APIs are crucial for facilitating communication between the frontend and the backend, allowing users to browse products, place orders, and manage their accounts seamlessly.
- **Security**: Security is paramount in any e-commerce platform. Node.js and Express.js provide robust libraries for implementing security measures such as authentication and validation. Notably, the use of JSON Web Tokens (JWT) for user authentication enhances the platform's security.
- **Performance**: The asynchronous and non-blocking nature of Node.js makes it highly performant, ensuring rapid processing of HTTP requests and responses. This performance is critical for maintaining a smooth user experience.

Frontend Client Layer: React.js

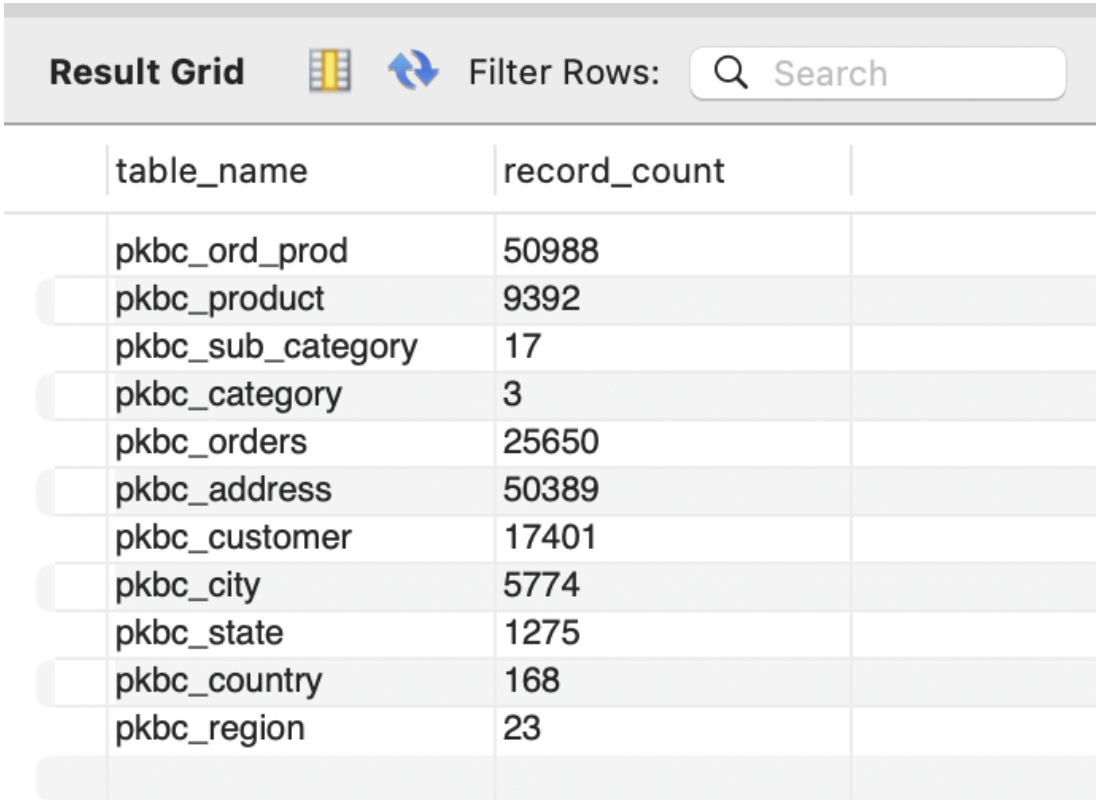
The frontend client layer utilizes React.js, a JavaScript library for building user interfaces. React.js is an excellent choice for Awesome Superstore Inc.'s web application due to these reasons:

- **User Interface**: React.js excels in creating dynamic and responsive user interfaces. Its component-based architecture allows for the development of reusable UI elements, a crucial factor in building a consistent and visually appealing online shopping experience.
- **State Management**: Managing the state of a web application is a complex task, especially for an e-commerce platform. React.js, combined with libraries like Redux, offers efficient state management, ensuring that user interactions and data updates are handled smoothly.
- **Component Reusability**: React.js promotes the creation of reusable UI components, enhancing development efficiency. Awesome Superstore Inc. has capitalized on this feature by building 12 distinct pages for the web application.
- **Routing**: The use of React Router for client-side routing simplifies navigation within the web application. This ensures that users can effortlessly explore product listings, view details, and navigate through the site.
- **Material UI**: Leveraging the Material UI component library contributes to a modern and visually appealing user interface. It enhances the platform's aesthetics and user experience.
- **Integration with Backend**: Axios, a networking tool, facilitates seamless communication between the React.js frontend and the Express.js backend, ensuring that data flows efficiently between the client and server.

I. Record counts for each OLTP tables

```
SELECT
  'pkbc_ord_prod' AS table_name,
  COUNT(*) AS record_count
FROM
  pkbc_ord_prod
UNION ALL
SELECT
  'pkbc_product' AS table_name,
  COUNT(*) AS record_count
FROM
  pkbc_product
UNION ALL
SELECT
  'pkbc_sub_category' AS table_name,
  COUNT(*) AS record_count
FROM
  pkbc_sub_category
UNION ALL
SELECT
  'pkbc_category' AS table_name,
  COUNT(*) AS record_count
FROM
  pkbc_category
UNION ALL
SELECT
  'pkbc_orders' AS table_name,
  COUNT(*) AS record_count
FROM
  pkbc_orders
UNION ALL
SELECT
  'pkbc_address' AS table_name,
```

```
  COUNT(*) AS record_count
FROM
  pkbc_address
UNION ALL
SELECT
  'pkbc_customer' AS table_name,
  COUNT(*) AS record_count
FROM
  pkbc_customer
UNION ALL
SELECT
  'pkbc_city' AS table_name,
  COUNT(*) AS record_count
FROM
  pkbc_city
UNION ALL
SELECT
  'pkbc_state' AS table_name,
  COUNT(*) AS record_count
FROM
  pkbc_state
UNION ALL
SELECT
  'pkbc_country' AS table_name,
  COUNT(*) AS record_count
FROM
  pkbc_country
UNION ALL
SELECT
  'pkbc_region' AS table_name,
  COUNT(*) AS record_count
FROM
  pkbc_region;
```



table_name	record_count
pkbc_ord_prod	50988
pkbc_product	9392
pkbc_sub_category	17
pkbc_category	3
pkbc_orders	25650
pkbc_address	50389
pkbc_customer	17401
pkbc_city	5774
pkbc_state	1275
pkbc_country	168
pkbc_region	23

J. Web application summary

The web application developed for Awesome Superstore Inc. represents a sophisticated and user-centric platform that seamlessly integrates cutting-edge technologies to provide an exceptional online shopping experience. The features in terms of engineering and the application functionality can be described below:

Engineering Features:

1. ReactJS & Material UI: Deliver a modern, visually appealing, and highly interactive user interface, enhancing user experience and engagement.
2. Redux & Local Storage: Efficiently manage application state, ensuring fluid user interactions, and seamless navigation.

3. React Router: Facilitate smooth page transitions across 12 unique pages, improving usability.
4. Security Measures: Employ BcryptJS, Express validator, and JWT for robust data protection, secure authentication, and password encryption, ensuring user data remains safe.
5. Database Indexing: Optimize data retrieval with strategically placed indexes, enhancing overall system performance.
6. ETL Triggers: Capture data update timestamps for ETL processes, facilitating data management.
7. Transaction Management: Safeguard data integrity with commit and rollback mechanisms, ensuring reliable database operations.

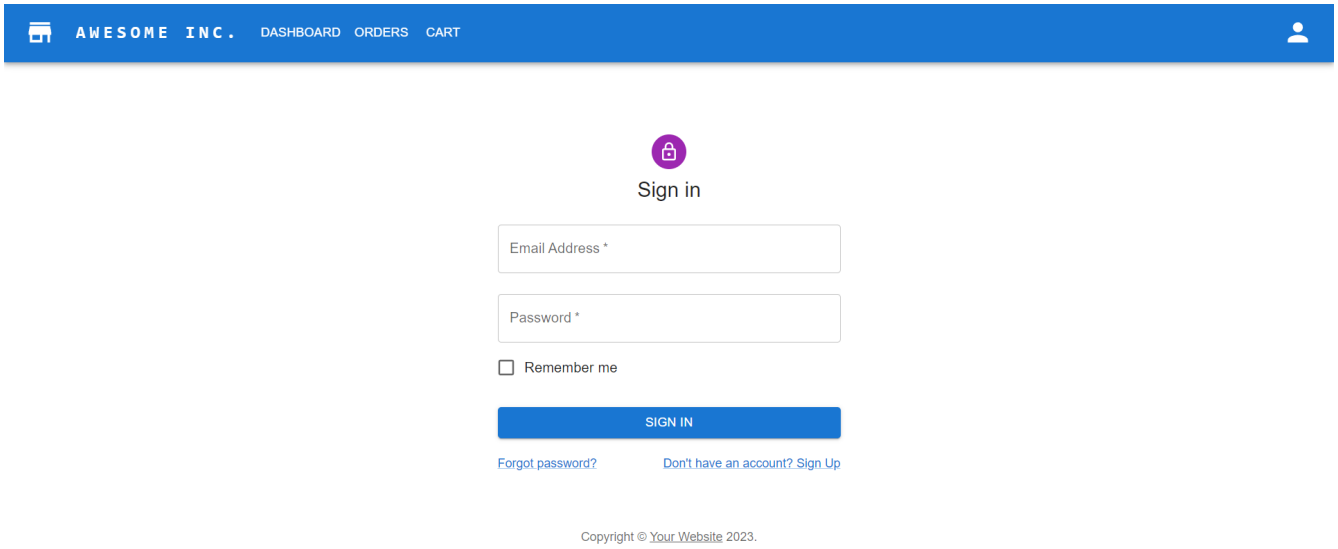
Application Functionality:

1. User Management: Streamlined account creation and OTP-based password reset for enhanced security.
2. Product Discovery: Easy product exploration and search capabilities, offering a wide product range.
3. Profile Customization: Personalize user profiles and manage passwords conveniently.
4. Address Management: Simplified address addition and editing for seamless ordering.
5. Shopping Cart: User-friendly product selection, quantity adjustments, and smooth order completion.
6. Order History: Transparent access to past orders with convenient filtering options.
7. Return Requests: Effortless return initiation and in-depth order insights for user convenience.
8. Admin Functionality: Admin can add new products in different markets to increase product availability.

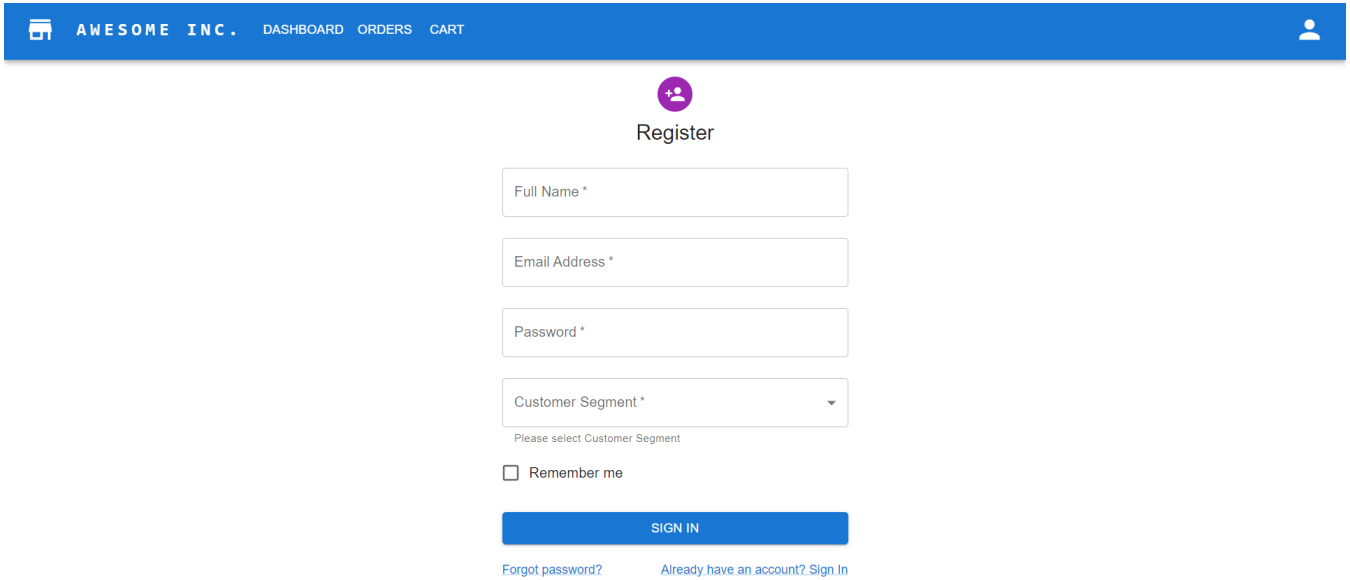
In summary, Awesome Superstore Inc.'s web application leverages modern engineering technologies to provide users with a secure, feature-rich, and highly interactive e-commerce platform. Notable features such as ReactJS, Redux, and robust security measures contribute to an exceptional user experience, while intuitive functionality streamlines the shopping process for customers.

K. Web application screenshots

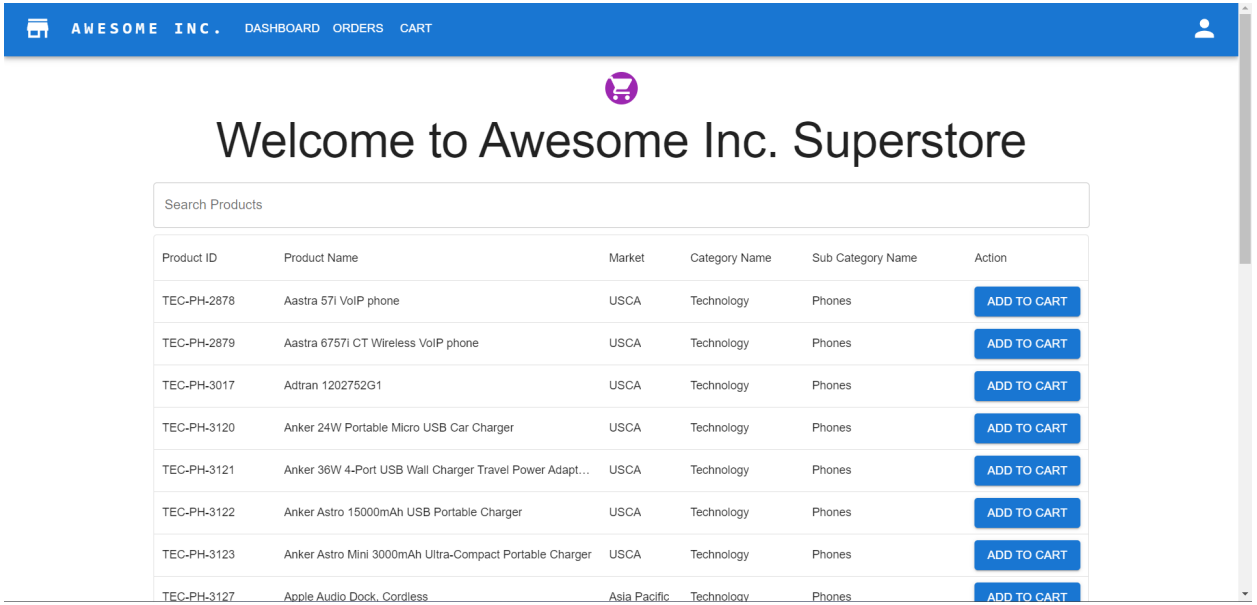
Login page



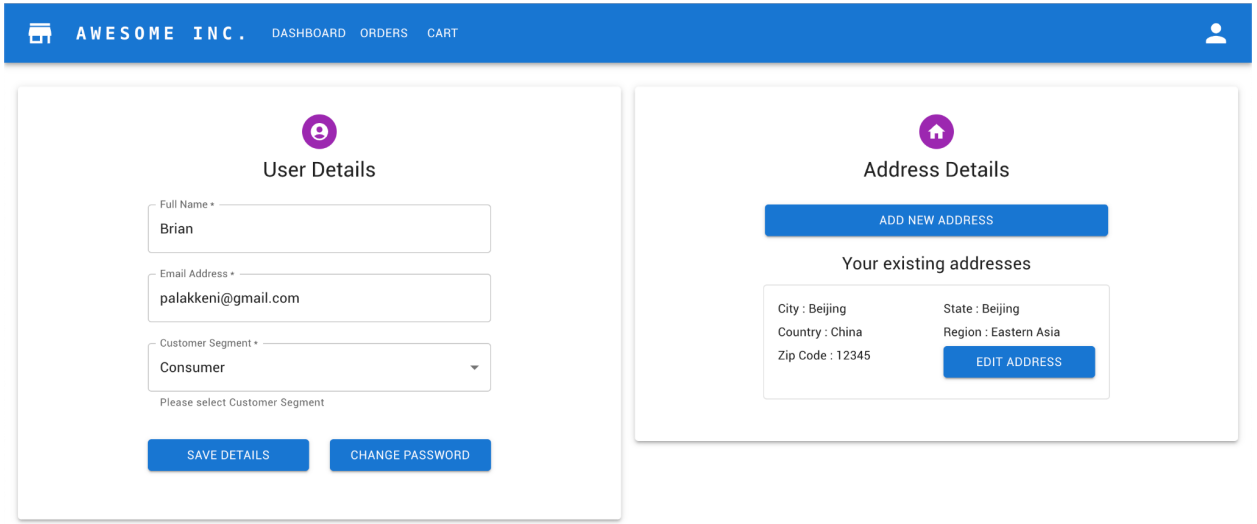
Register page


AWESOME INC.
DASHBOARD
ORDERS
CART
Register
Full Name *
Email Address *
Password *
Customer Segment *
Please select Customer Segment
Remember me
SIGN IN
Forgot password?
Already have an account? Sign In

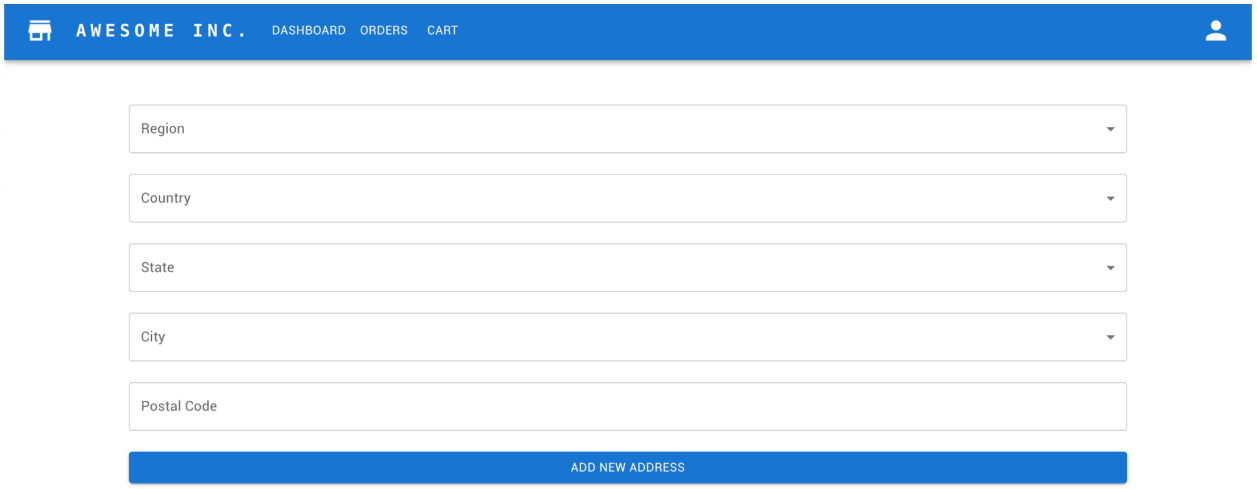
Dashboard



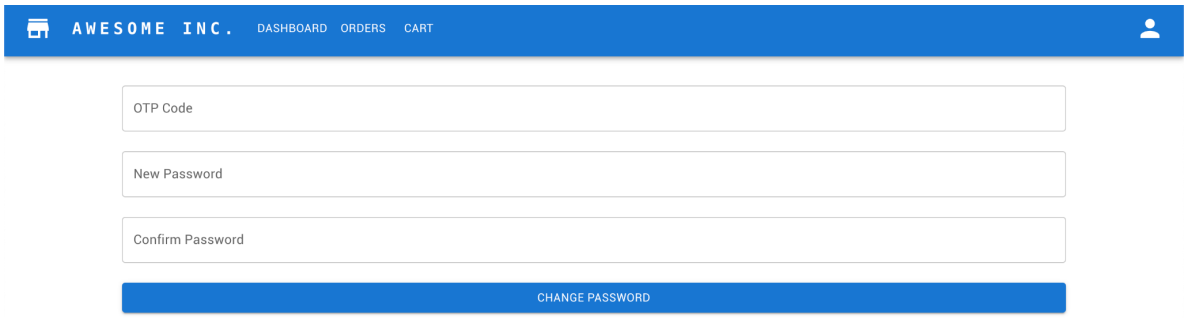
User Profile



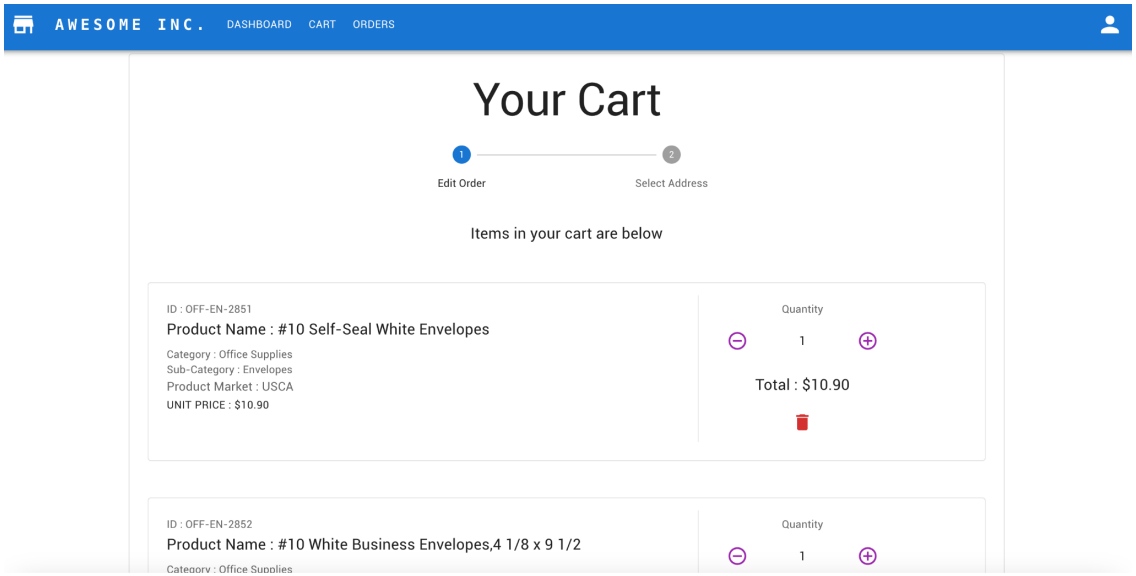
Add/Edit Address



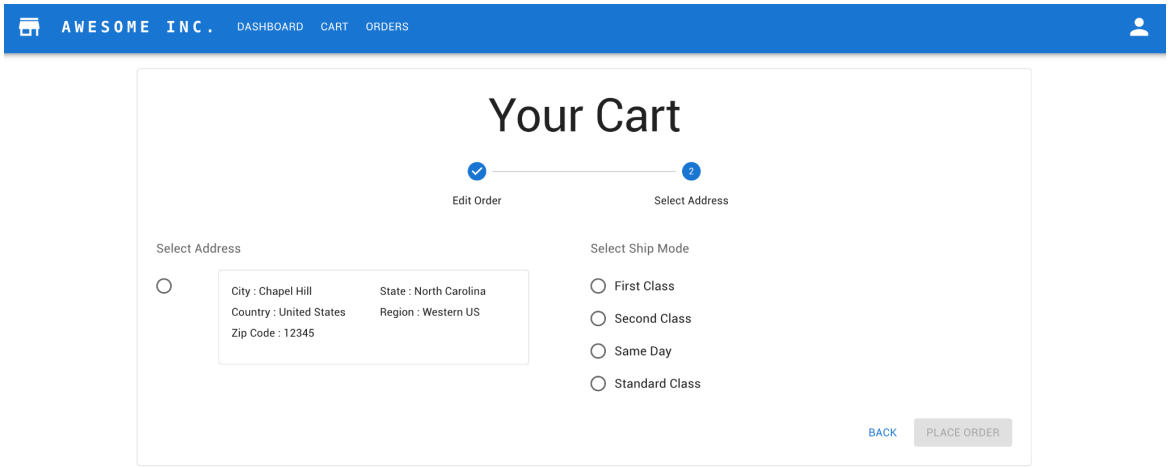
Change Password



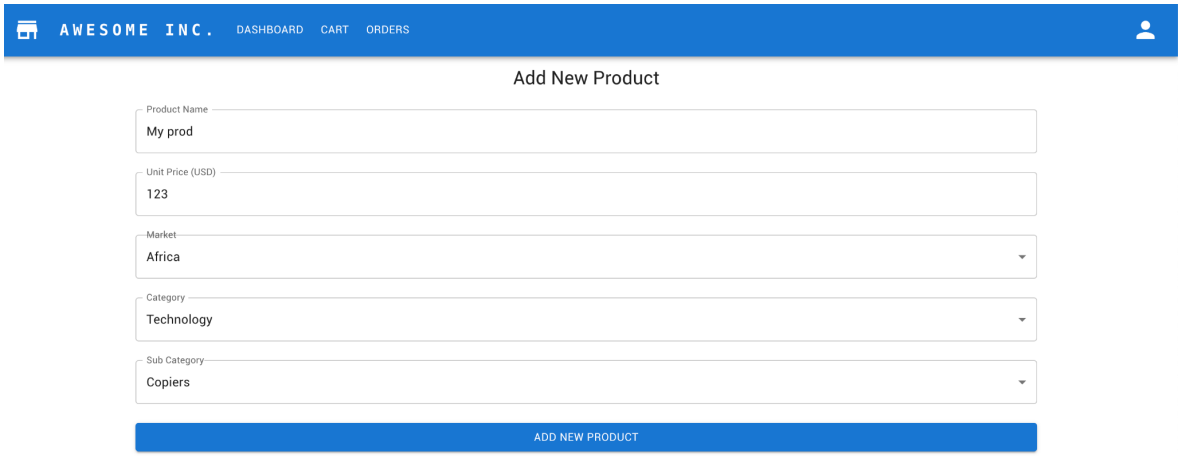
Cart



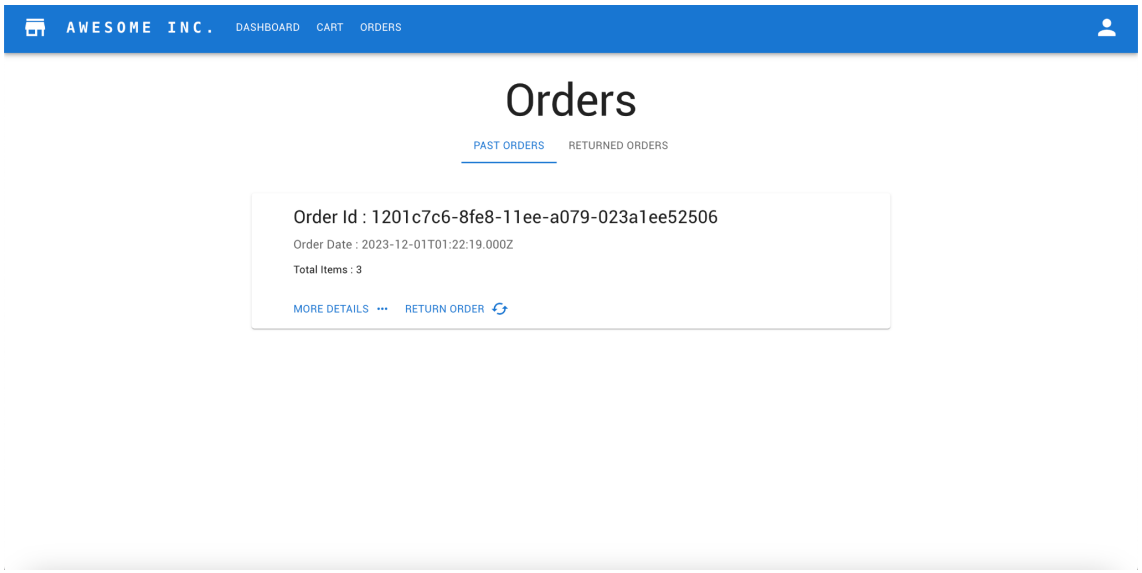
Place Order



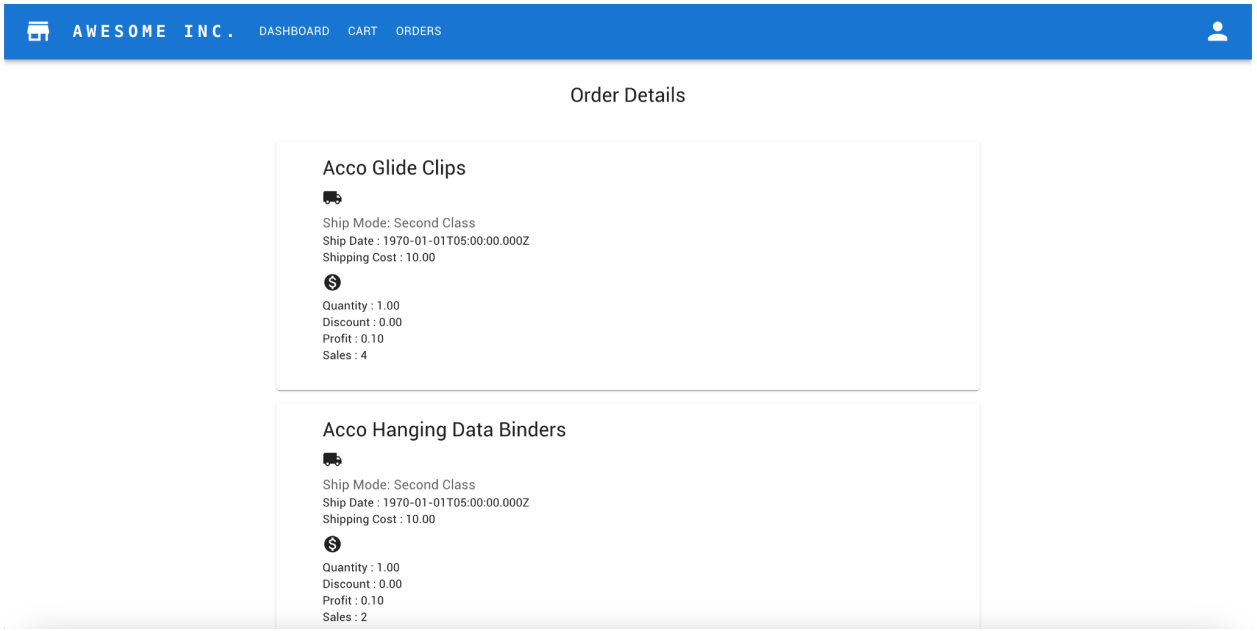
Add Product



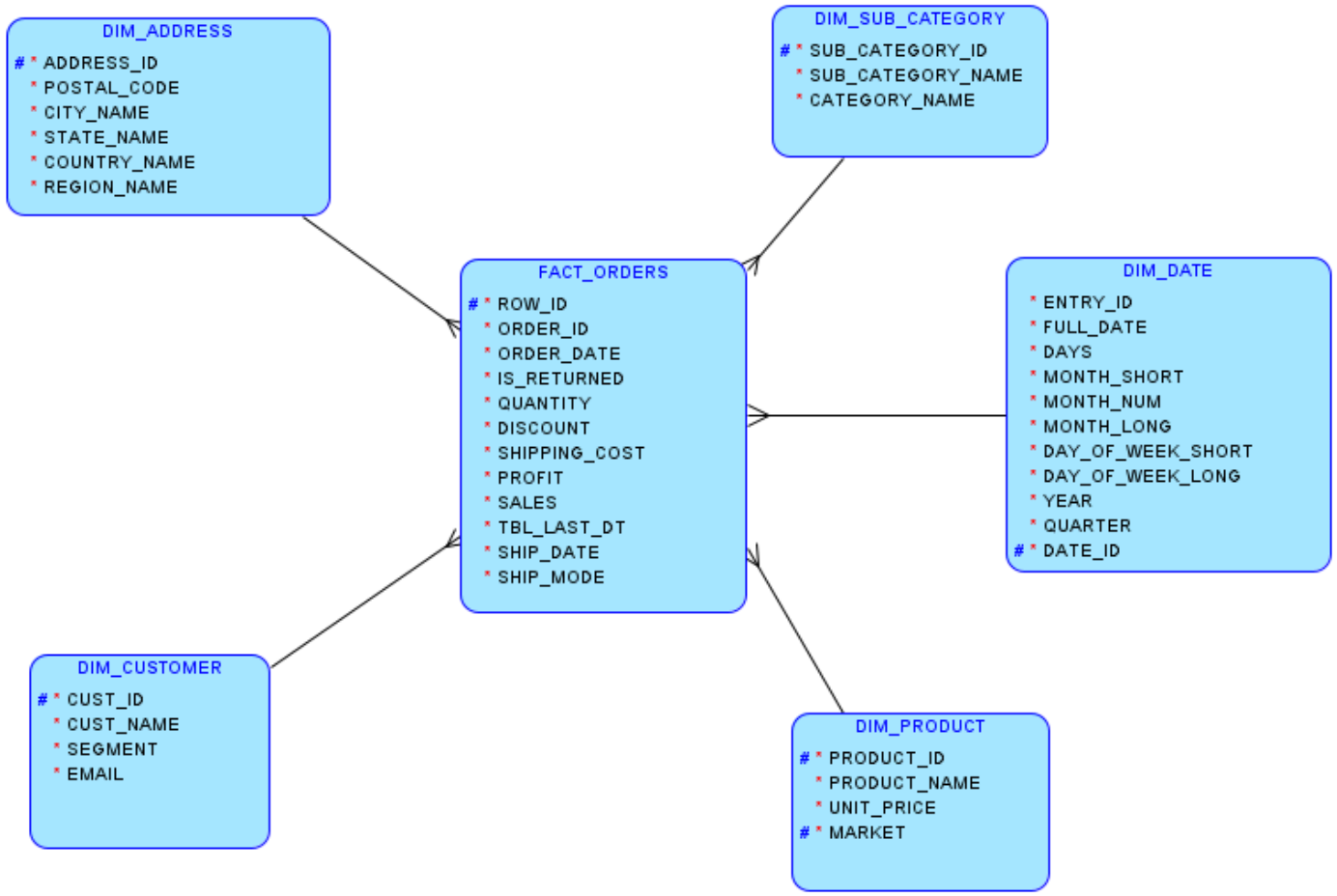
Past Orders



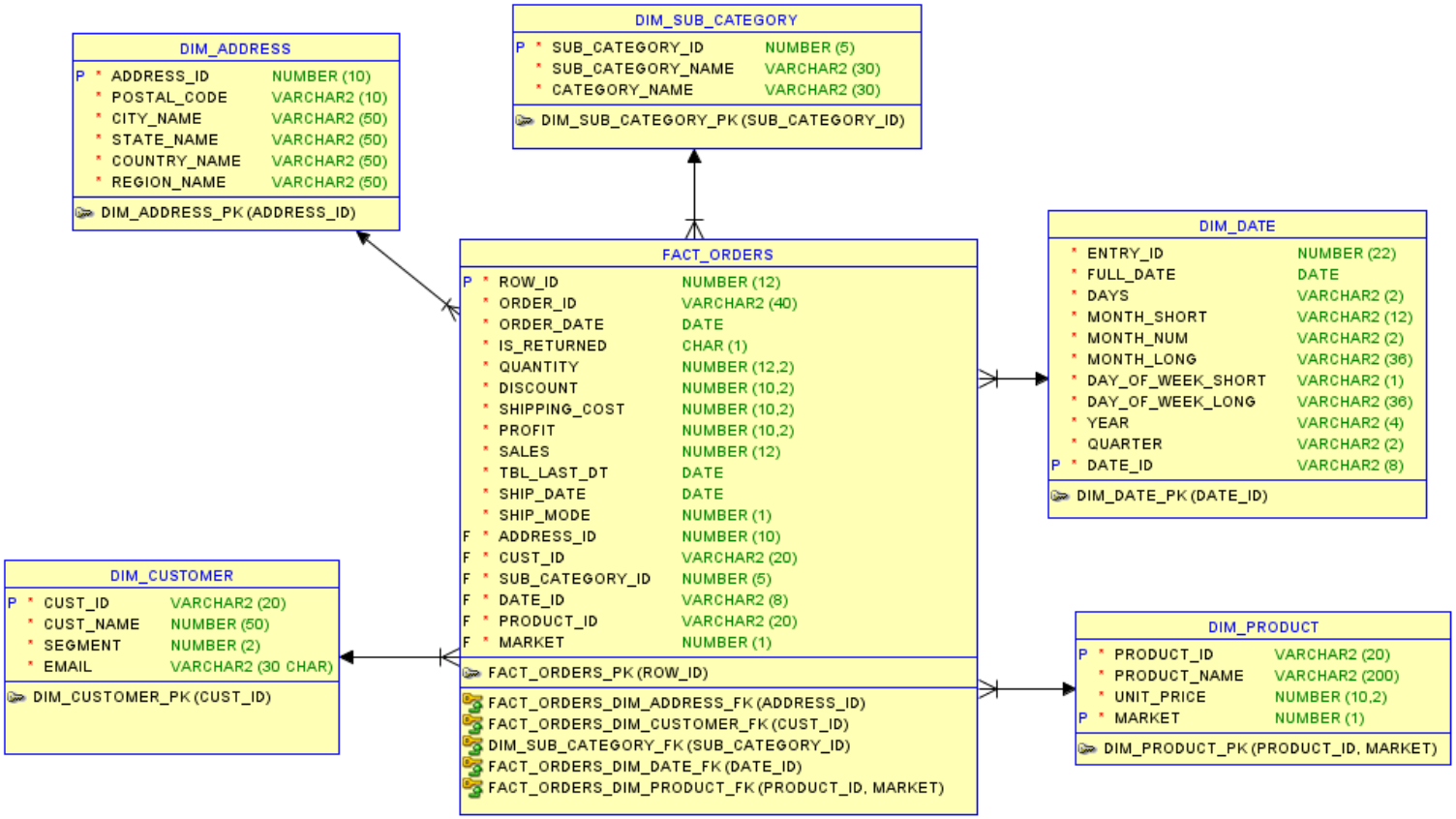
Order Details



L. Logical Model (DW)



M. Relational Model(DW)



N. Summary of ETL approach

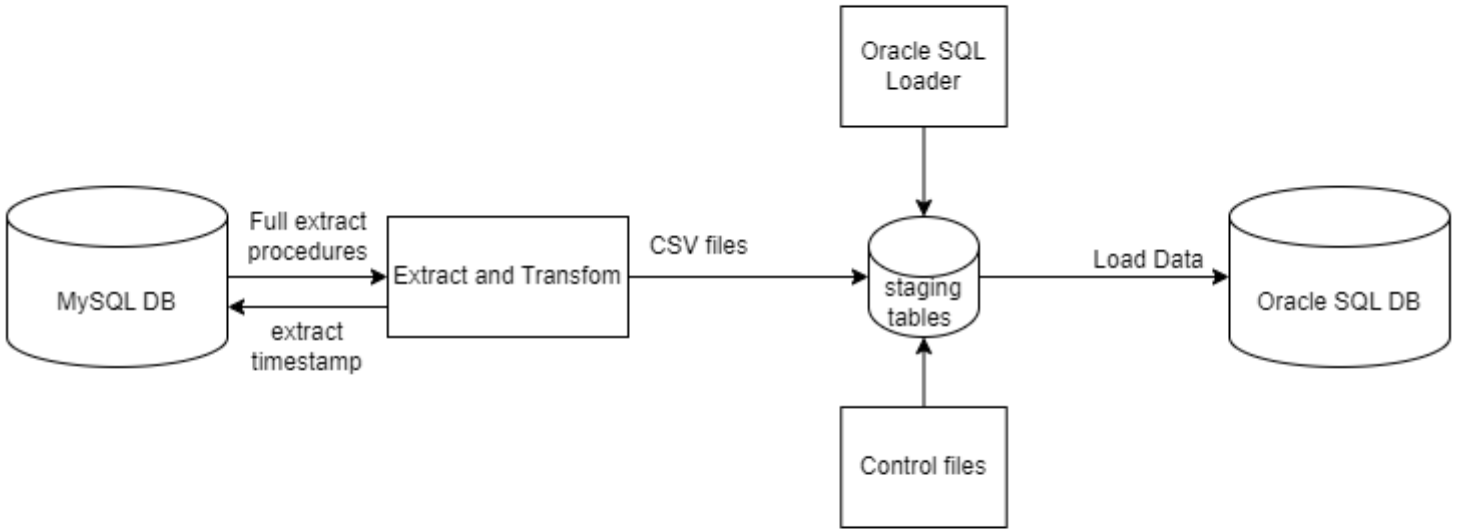
In this project we have used the ETL approach to integrate data from OLTP database(MySQL DB) to a data warehouse(Oracle SQL DB). We have used the CDC (Change data capture) process to perform the ETL activity on a scheduled basis. This enables keeping the data warehouse updated efficiently through minimized data transfers. The data was extracted in the form of .csv files and then exported to Oracle SQL using the SQL loader.

We have used the star schema design to implement the data warehouse as it is an industrial standard and is useful for its simplicity and fast querying. In addition to the data warehouse schema tables, we have used staging tables which are used to stage data before inserting into the data warehouse dimensional and fact tables. As mentioned in the diagram above (Relational and Logical), we have 5 dimension tables and 1 fact table, and there are 6 staging tables corresponding to these tables. Since data integration was done using the CDC method, we have 2 steps in the process.

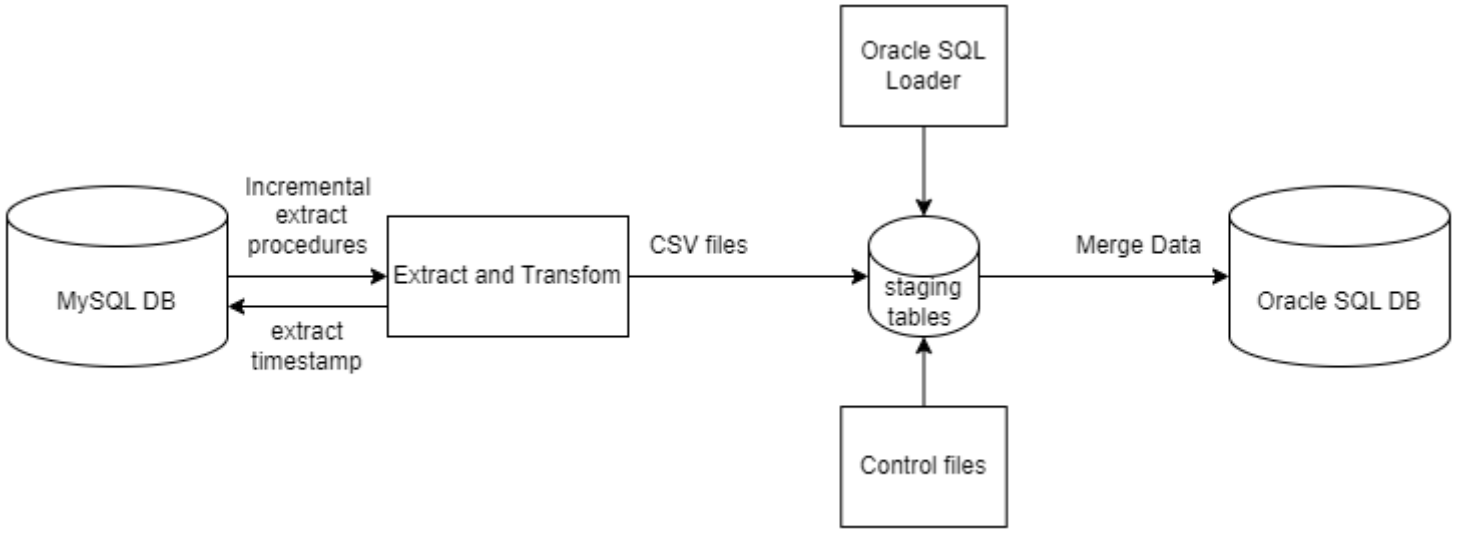
In the first step we have a Full ETL process. For this initial process, full extracts were taken from the MySQL OLTP database tables in the form of CSV files. We have written a MySQL procedure to capture all the current data inside the MySQL database and transform the data in the extract process itself to be used for data warehouse staging. Additionally, we have created a table **(etl_extract_date)** in which we are storing the timestamp of the successfully completed full ETL extract. These CSV extracts were then transformed and loaded into the staging tables in Oracle using SQL Loader. After applying business logic and transformations using Oracle SQL procedures, the staged data was further loaded into the production data warehouse tables. This process established the full dataset inside the Oracle data warehouse to which the subsequent incremental loads have to sync to.

The second part of this process was the Incremental ETL process. For this next process, we have captured data into .csv files from the MySQL database which was inserted after the latest extract timestamp stored in the etl_extract_date table. Similar to the full ETL process, we have written a MySQL procedure to capture the all changed data from the latest timestamp and transform the data in the extract process itself to be used for data warehouse staging. Additionally, we are inserting a new entry in the etl_extract_date table in which we are storing the timestamp of the successfully completed incremental extract. These CSV extracts were then transformed and loaded into the staging tables in Oracle using SQL Loader. After applying business logic and transformations using Oracle SQL procedures, the staged data was merged into the production data warehouse tables.This incremental process enables synchronizing the data warehouse with the OLTP database by transferring and updating only the changed records after the one-time full load.

FULL ETL PROCESS



INCREMENTAL ETL PROCESS



O. Analysis from DW systems

SQL count data in DW

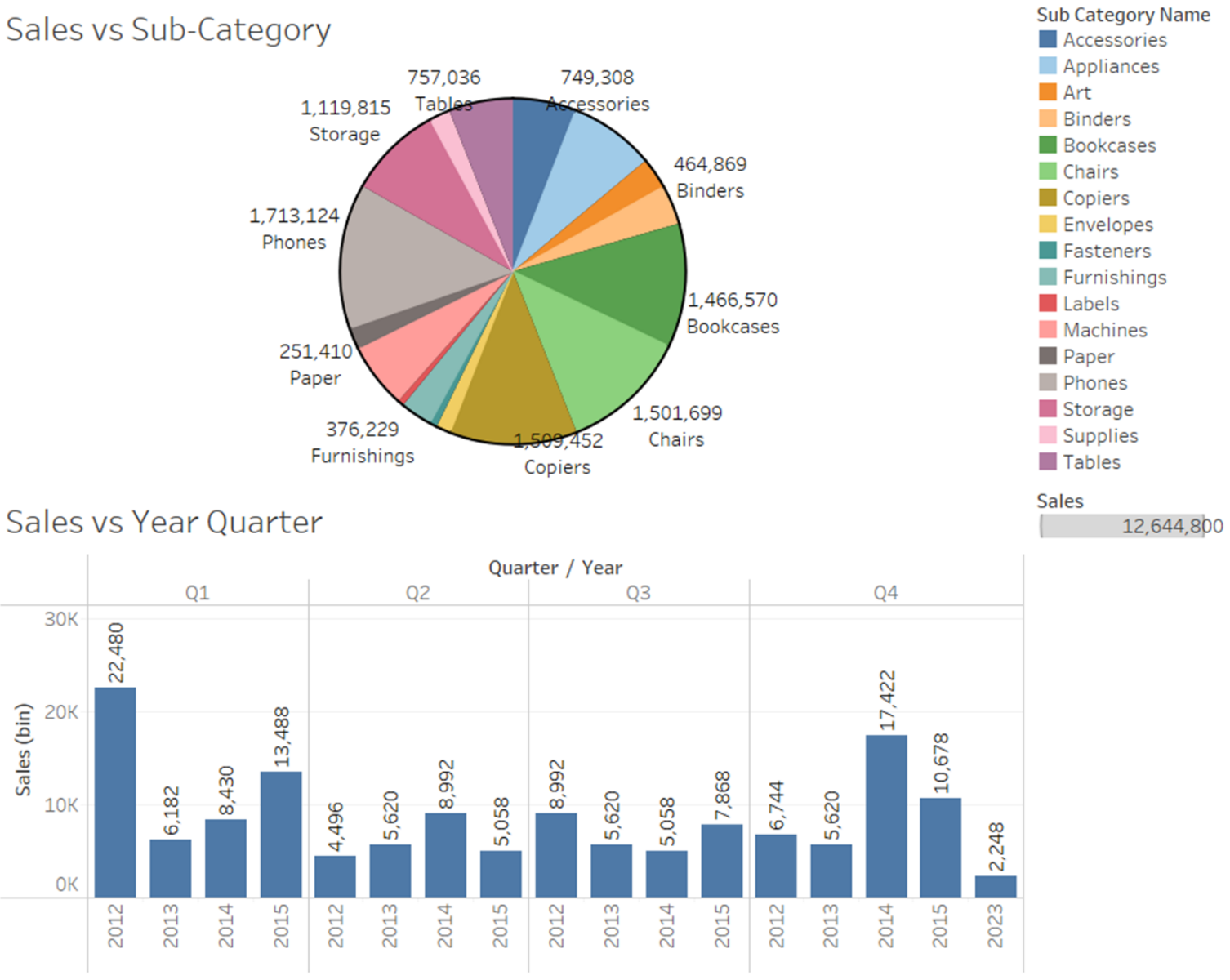
```
20  SELECT TABLE_NAME,
21        to_number( extractvalue( xmltype(
22  dbms_xmlgen.getxml('SELECT COUNT(*) c FROM '|| table_name))  ,'/ROWSET/ROW/C')) AS "ROW COUNT"
23      FROM USER_TABLES;
```

Query Result

SQL | All Rows Fetched: 11 in 0.371 seconds

	TABLE_NAME	ROW COUNT
1	STG_PKBC_DIM_SUB_CATEGORY	17
2	DIM_PKBC_ADDRESS	51191
3	DIM_PKBC_CUSTOMER	17417
4	DIM_PKBC_DATE	36600
5	DIM_PKBC_PRODUCT	9447
6	DIM_PKBC_SUB_CATEGORY	17
7	STG_PKBC_DIM_CUSTOMER	17417
8	STG_PKBC_DIM_PRODUCT	9447
9	STG_PKBC_FACT_ORDERS	51095
10	STG_PKBC_DIM_ADDRESS	51191
11	FACT_PKBC_ORDERS	51095

Tableau dashboard:



P. Lesson learned

This project in creating the web application, ETL, data warehousing and analytics provided some great lessons for our computer science career:

1. Working with a partner, we realized that we each had different skills. One was good at developing the backend, while the other excelled at creating the user interface (UI). We learned that by combining our strengths, we could cover each other's weaknesses. This collaboration made our project more well-rounded.

2. To work effectively, we needed to communicate well. We had to make sure our work didn't clash or cause problems. Regular discussions helped us understand each other's needs, make design choices, and overcome challenges. Good communication was essential for our partnership to succeed.

3. We found that tools like Git and GitHub were incredibly useful. They allowed us to manage our code and work together seamlessly. These tools helped us keep track of changes, collaborate without issues, and fix conflicts. Git's features allowed us to work on different parts of the project simultaneously, which made us more productive.

4. We learned that writing clean and easy-to-understand code was crucial. This wasn't just about personal preferences; it was a necessity. Clean code made it easier for our partner to read, review, and contribute to the project. It also made finding and fixing problems simpler and reduced errors.

Q. Appendix

Code repository is located on github for the following

1. Front end: https://github.com/briancatraguna/awesome-superstore-frontend
2. Back end: https://github.com/briancatraguna/awesome-superstore-backend
3. OLTP and DW scripts: https://github.com/palakkeni5/awesome_superetore_db

a. **OLTP DDL code (Tables, Constraints, Trigger, History tables, any function/ procedure created etc.)**

DATABASE SCHEMA CREATION, CONSTRAINTS AND INDEXES

```
create schema awesome_inc;
use awesome_inc;

CREATE TABLE pkbc_address (
    addr_id     BIGINT PRIMARY KEY AUTO_INCREMENT COMMENT 'Address ID',
    city_id     BIGINT NOT NULL COMMENT 'City id',
    postal_code VARCHAR(10) COMMENT 'Postal code',
    cust_id     VARCHAR(20) NOT NULL COMMENT 'Customer Id ',
    tbl_last_dt DATETIME NOT NULL COMMENT 'Timestamp for the row data added'
);

CREATE INDEX cust_id_idx ON pkbc_address (cust_id);


CREATE TABLE pkbc_sub_category (
    sub_category_id       INT NOT NULL PRIMARY KEY AUTO_INCREMENT COMMENT 'Sub Category id.',
    sub_category_name     VARCHAR(30) NOT NULL COMMENT 'Product Sub Category Name',
    category_id       INT NOT NULL COMMENT 'Category id.',
    tbl_last_dt       DATETIME NOT NULL COMMENT 'Timestamp for the row data added'
);
CREATE TABLE pkbc_category (
    category_id       INT NOT NULL PRIMARY KEY AUTO_INCREMENT COMMENT 'Category id.',
    category_name     VARCHAR(30) NOT NULL COMMENT 'Product Category Name',
    tbl_last_dt       DATETIME NOT NULL COMMENT 'Timestamp for the row data added'
);
```

```sql
CREATE TABLE pkbc_city (
    city_id     BIGINT NOT NULL PRIMARY KEY AUTO_INCREMENT COMMENT 'City Id',
    city_name   VARCHAR(50) NOT NULL COMMENT 'City Name',
    state_id    BIGINT NOT NULL COMMENT 'State id',
    tbl_last_dt DATETIME NOT NULL COMMENT 'Timestamp for the row data added'
);
```

```sql
CREATE TABLE pkbc_country (
    country_id   BIGINT NOT NULL PRIMARY KEY AUTO_INCREMENT COMMENT 'Country id',
    country_name VARCHAR(50) NOT NULL COMMENT 'Country Name',
    region_id    BIGINT NOT NULL COMMENT 'Region id',
    tbl_last_dt  DATETIME NOT NULL COMMENT 'Timestamp for the row data added'
);
```

```sql
CREATE TABLE pkbc_customer (
    cust_id     VARCHAR(20) NOT NULL COMMENT 'Customer Id.',
    cust_name   VARCHAR(100) NOT NULL COMMENT 'Customer name',
    segment     TINYINT NOT NULL COMMENT 'Customer Segment. 1: Consumer , 2: Corporate, 3: Home
Office',
    email       VARCHAR(30) NOT NULL COMMENT 'Email of the customer',
    `password`  VARCHAR(100) NOT NULL COMMENT 'Password of the login of the customer',
    tbl_last_dt DATETIME NOT NULL COMMENT 'Timestamp for the row data added'
);
```

```
ALTER TABLE pkbc_customer ADD CONSTRAINT pkbc_customer_pk PRIMARY KEY ( cust_id );

CREATE TABLE pkbc_ord_prod (
      ord_prod_id   BIGINT  NOT NULL PRIMARY KEY AUTO_INCREMENT COMMENT 'unique id for all orders
and products',
    quantity      DECIMAL(12, 2) NOT NULL COMMENT 'Product Quantity for the order',
    discount      DECIMAL(10, 2) NOT NULL COMMENT 'Discount for product of  the order',
    shipping_cost DECIMAL(10, 2) NOT NULL COMMENT 'Shipping cost for product of  the order',
    profit        DECIMAL(10, 2) NOT NULL COMMENT 'Profit for product of  the order',
    sales         BIGINT NOT NULL COMMENT 'Sales for the product of the order',
    order_id      VARCHAR(40) NOT NULL COMMENT 'Order Id',
    product_id    VARCHAR(20) NOT NULL COMMENT 'Product Id',
    tbl_last_dt   DATETIME NOT NULL COMMENT 'Timestamp for the row data added',
    market        TINYINT NOT NULL,
    addr_id       BIGINT NOT NULL,
    cust_id       VARCHAR(20) NOT NULL COMMENT 'Customer Id',
    ship_mode     TINYINT NOT NULL COMMENT 'shipping  mode. 1: First Class , 2 : Second Class ,
3 : Same Day, 4 : Standard Class',
    ship_date     DATETIME NOT NULL COMMENT 'Shipping Date'
);

CREATE INDEX cust_id_idx ON pkbc_ord_prod (cust_id);
CREATE INDEX product_id_idx ON pkbc_ord_prod (product_id);


CREATE TABLE pkbc_orders (
    order_id    VARCHAR(40) NOT NULL COMMENT 'Order Id',
    order_date  DATETIME NOT NULL COMMENT 'Order Date',
    is_returned CHAR(1) NOT NULL COMMENT 'Is order returned. True : returned, False: not
```

```
returned',
    tbl_last_dt DATETIME NOT NULL COMMENT 'Timestamp for the row data added'
);


ALTER TABLE pkbc_orders ADD CONSTRAINT pkbc_orders_pk PRIMARY KEY ( order_id );

CREATE TABLE pkbc_product (
    product_id   VARCHAR(20) NOT NULL COMMENT 'Product  id.',
    product_name VARCHAR(200) NOT NULL COMMENT 'Product Name',
    market       TINYINT NOT NULL COMMENT 'Product Market. 1: Africa, 2: Asia Pacific, 3:
Europe, 4: LATAM, 5: USCA',
    sub_category_id  INT NOT NULL COMMENT 'Category Id',
    tbl_last_dt  DATETIME NOT NULL COMMENT 'Timestamp for the row data added'
);

CREATE INDEX product_id_idx ON pkbc_product (product_id);

ALTER TABLE pkbc_product ADD CONSTRAINT pkbc_product_pk PRIMARY KEY ( product_id,
                                                                      market );

CREATE TABLE pkbc_region (
    region_id   BIGINT NOT NULL  PRIMARY KEY AUTO_INCREMENT COMMENT 'Country id',
    region_name VARCHAR(50) NOT NULL COMMENT 'Region  Name',
    tbl_last_dt DATETIME NOT NULL COMMENT 'Timestamp for the row data added'
);


CREATE TABLE pkbc_state (
    state_id    BIGINT NOT NULL  PRIMARY KEY AUTO_INCREMENT COMMENT 'state id',
```

```
    state_name  VARCHAR(50) NOT NULL COMMENT 'State Name',
    country_id   BIGINT NOT NULL COMMENT 'Country id',
    tbl_last_dt DATETIME NOT NULL COMMENT 'Timestamp for the row data added'
);



ALTER TABLE pkbc_address
    ADD CONSTRAINT pkbc_address_pkbc_city_fk FOREIGN KEY ( city_id )
        REFERENCES pkbc_city ( city_id );

ALTER TABLE pkbc_address
    ADD CONSTRAINT pkbc_address_pkbc_customer_fk FOREIGN KEY ( cust_id )
        REFERENCES pkbc_customer ( cust_id )
        ON UPDATE CASCADE;

ALTER TABLE pkbc_country
     ADD CONSTRAINT pkbc_country_pkbc_region_fk FOREIGN KEY ( region_id )
          REFERENCES pkbc_region ( region_id );

ALTER TABLE pkbc_state
     ADD CONSTRAINT pkbc_state_pkbc_country_fk FOREIGN KEY ( country_id )
          REFERENCES pkbc_country ( country_id );

ALTER TABLE pkbc_city
     ADD CONSTRAINT pkbc_city_pkbc_state_fk FOREIGN KEY ( state_id )
          REFERENCES pkbc_state ( state_id );

ALTER TABLE pkbc_ord_prod
    ADD CONSTRAINT pkbc_ord_prod_pkbc_address_fk FOREIGN KEY ( addr_id )
```

```
        REFERENCES pkbc_address ( addr_id );

ALTER TABLE pkbc_ord_prod
    ADD CONSTRAINT pkbc_ord_prod_pkbc_orders_fk FOREIGN KEY ( order_id )
        REFERENCES pkbc_orders ( order_id );

ALTER TABLE pkbc_ord_prod
    ADD CONSTRAINT pkbc_ord_prod_pkbc_product_fk FOREIGN KEY ( product_id,
                                                               market )
        REFERENCES pkbc_product ( product_id,
                                  market );

ALTER TABLE pkbc_ord_prod
    ADD CONSTRAINT pkbc_ord_prod_pkbc_customer_fk FOREIGN KEY ( cust_id )
        REFERENCES pkbc_customer ( cust_id )
        ON UPDATE CASCADE;

ALTER TABLE pkbc_product
    ADD CONSTRAINT pkbc_sub_category_fk FOREIGN KEY ( sub_category_id )
        REFERENCES pkbc_sub_category ( sub_category_id );

ALTER TABLE pkbc_sub_category
    ADD CONSTRAINT pkbc_category_fk FOREIGN KEY ( category_id )
        REFERENCES pkbc_category ( category_id );

ALTER TABLE pkbc_customer
ADD otp_code VARCHAR(5);


ALTER TABLE pkbc_product
```

```
ADD COLUMN unit_price DECIMAL(10,2) NOT NULL DEFAULT 0;
```

TIME COLUMN TRIGGERS

```
DROP TRIGGER IF EXISTS TI_address_default_date;

DELIMITER $$
CREATE TRIGGER `TI_address_default_date`
      BEFORE INSERT ON `pkbc_address`
      FOR EACH ROW
      BEGIN
            if ( isnull(new.tbl_last_dt) ) then
                  set new.tbl_last_dt=current_timestamp();
            end if;
      END$$;
DELIMITER ;

DROP TRIGGER IF EXISTS TU_address_default_date;


DELIMITER $$
CREATE TRIGGER `TU_address_default_date`
      BEFORE UPDATE ON `pkbc_address`
      FOR   EACH ROW
      BEGIN
            set NEW.tbl_last_dt=current_timestamp();
      END$$;
```

```
DELIMITER ;

-- pkbc_category TABLE TRIGGER

DROP TRIGGER IF EXISTS TI_category_default_date;

DELIMITER $$
CREATE TRIGGER `TI_category_default_date`
      BEFORE INSERT ON `pkbc_category`
      FOR EACH ROW
      BEGIN
            if ( isnull(new.tbl_last_dt) ) then
                  set new.tbl_last_dt=current_timestamp();
            end if;
      END$$;
DELIMITER ;

DROP TRIGGER IF EXISTS TU_category_default_date;


DELIMITER $$
CREATE TRIGGER `TU_category_default_date`
      BEFORE UPDATE ON `pkbc_category`
      FOR   EACH ROW
      BEGIN
            set NEW.tbl_last_dt=current_timestamp();
      END$$;
DELIMITER ;

-- pkbc_sub_category TABLE TRIGGER
```

```
DROP TRIGGER IF EXISTS TI_sub_category_default_date;

DELIMITER $$
CREATE TRIGGER `TI_sub_category_default_date`
      BEFORE INSERT ON `pkbc_sub_category`
      FOR EACH ROW
      BEGIN
            if ( isnull(new.tbl_last_dt) ) then
                  set new.tbl_last_dt=current_timestamp();
            end if;
      END$$;
DELIMITER ;

DROP TRIGGER IF EXISTS TU_sub_category_default_date;


DELIMITER $$
CREATE TRIGGER `TU_sub_category_default_date`
      BEFORE UPDATE ON `pkbc_sub_category`
      FOR   EACH ROW
      BEGIN
            set NEW.tbl_last_dt=current_timestamp();
      END$$;
DELIMITER ;



-- pkbc_city TABLE TRIGGER
```

```
DROP TRIGGER IF EXISTS TI_city_default_date;

DELIMITER $$
CREATE TRIGGER `TI_city_default_date`
      BEFORE INSERT ON `pkbc_city`
      FOR EACH ROW
      BEGIN
            if ( isnull(new.tbl_last_dt) ) then
                  set new.tbl_last_dt=current_timestamp();
            end if;
      END$$;
DELIMITER ;

DROP TRIGGER IF EXISTS TU_city_default_date;


DELIMITER $$
CREATE TRIGGER `TU_city_default_date`
      BEFORE UPDATE ON `pkbc_city`
      FOR   EACH ROW
      BEGIN
            set NEW.tbl_last_dt=current_timestamp();
      END$$;
DELIMITER ;

-- pkbc_country TABLE TRIGGER

DROP TRIGGER IF EXISTS TI_country_default_date;

DELIMITER $$
```

```
CREATE TRIGGER `TI_country_default_date`
      BEFORE INSERT ON `pkbc_country`
      FOR EACH ROW
      BEGIN
            if ( isnull(new.tbl_last_dt) ) then
                  set new.tbl_last_dt=current_timestamp();
            end if;
      END$$;
DELIMITER ;

DROP TRIGGER IF EXISTS TU_country_default_date;


DELIMITER $$
CREATE TRIGGER `TU_country_default_date`
      BEFORE UPDATE ON `pkbc_country`
      FOR   EACH ROW
      BEGIN
            set NEW.tbl_last_dt=current_timestamp();
      END$$;
DELIMITER ;

-- pkbc_customer TABLE TRIGGER

DROP TRIGGER IF EXISTS TI_customer_default_date;

DELIMITER $$
CREATE TRIGGER `TI_customer_default_date`
      BEFORE INSERT ON `pkbc_customer`
      FOR EACH ROW
```

```
      BEGIN
            if ( isnull(new.tbl_last_dt) ) then
                  set new.tbl_last_dt=current_timestamp();
            end if;
      END$$;
DELIMITER ;

DROP TRIGGER IF EXISTS TU_customer_default_date;


DELIMITER $$
CREATE TRIGGER `TU_customer_default_date`
      BEFORE UPDATE ON `pkbc_customer`
      FOR   EACH ROW
      BEGIN
            set NEW.tbl_last_dt=current_timestamp();
      END$$;
DELIMITER ;

-- pkbc_ord_prod TABLE TRIGGER

DROP TRIGGER IF EXISTS TI_ord_prod_default_date;

DELIMITER $$
CREATE TRIGGER `TI_ord_prod_default_date`
      BEFORE INSERT ON `pkbc_ord_prod`
      FOR EACH ROW
      BEGIN
            if ( isnull(new.tbl_last_dt) ) then
                  set new.tbl_last_dt=current_timestamp();
```

```
            end if;
      END$$;
DELIMITER ;

DROP TRIGGER IF EXISTS TU_ord_prod_default_date;


DELIMITER $$
CREATE TRIGGER `TU_ord_prod_default_date`
      BEFORE UPDATE ON `pkbc_ord_prod`
      FOR   EACH ROW
      BEGIN
            set NEW.tbl_last_dt=current_timestamp();
      END$$;
DELIMITER ;

-- pkbc_orders TABLE TRIGGER

DROP TRIGGER IF EXISTS TI_orders_default_date;

DELIMITER $$
CREATE TRIGGER `TI_orders_default_date`
      BEFORE INSERT ON `pkbc_orders`
      FOR EACH ROW
      BEGIN
            if ( isnull(new.tbl_last_dt) ) then
                  set new.tbl_last_dt=current_timestamp();
            end if;
      END$$;
DELIMITER ;
```

```
DROP TRIGGER IF EXISTS TU_orders_default_date;


DELIMITER $$
CREATE TRIGGER `TU_orders_default_date`
      BEFORE UPDATE ON `pkbc_orders`
      FOR   EACH ROW
      BEGIN
            set NEW.tbl_last_dt=current_timestamp();
      END$$;
DELIMITER ;

-- pkbc_product TABLE TRIGGER

DROP TRIGGER IF EXISTS TI_product_default_date;

DELIMITER $$
CREATE TRIGGER `TI_product_default_date`
      BEFORE INSERT ON `pkbc_product`
      FOR EACH ROW
      BEGIN
            if ( isnull(new.tbl_last_dt) ) then
                  set new.tbl_last_dt=current_timestamp();
            end if;
      END$$;
DELIMITER ;

DROP TRIGGER IF EXISTS TU_product_default_date;
```

```
DELIMITER $$
CREATE TRIGGER `TU_product_default_date`
      BEFORE UPDATE ON `pkbc_product`
      FOR   EACH ROW
      BEGIN
            set NEW.tbl_last_dt=current_timestamp();
      END$$;
DELIMITER ;

-- pkbc_region TABLE TRIGGER

DROP TRIGGER IF EXISTS TI_region_default_date;

DELIMITER $$
CREATE TRIGGER `TI_region_default_date`
      BEFORE INSERT ON `pkbc_region`
      FOR EACH ROW
      BEGIN
            if ( isnull(new.tbl_last_dt) ) then
                  set new.tbl_last_dt=current_timestamp();
            end if;
      END$$;
DELIMITER ;

DROP TRIGGER IF EXISTS TU_region_default_date;


DELIMITER $$
CREATE TRIGGER `TU_region_default_date`
```

```
      BEFORE UPDATE ON `pkbc_region`
      FOR   EACH ROW
      BEGIN
            set NEW.tbl_last_dt=current_timestamp();
      END$$;
DELIMITER ;

-- pkbc_state TABLE TRIGGER

DROP TRIGGER IF EXISTS TI_state_default_date;

DELIMITER $$
CREATE TRIGGER `TI_state_default_date`
      BEFORE INSERT ON `pkbc_state`
      FOR EACH ROW
      BEGIN
            if ( isnull(new.tbl_last_dt) ) then
                  set new.tbl_last_dt=current_timestamp();
            end if;
      END$$;
DELIMITER ;

DROP TRIGGER IF EXISTS TU_state_default_date;


DELIMITER $$
CREATE TRIGGER `TU_state_default_date`
      BEFORE UPDATE ON `pkbc_state`
      FOR   EACH ROW
      BEGIN
```

```
            set NEW.tbl_last_dt=current_timestamp();
      END$$;
DELIMITER ;
```

HISTORY TABLES

```
drop table if exists pkbc_address_history;
create table pkbc_address_history as select * from pkbc_address where 1 = 2;
alter table pkbc_address_history add constraint pk_address_history primary key (addr_id);

delimiter $$
drop trigger if exists td_pkbc_address;
create trigger td_pkbc_address
before delete on pkbc_address for each row
begin
insert into pkbc_address_history
select * from pkbc_address
where addr_id = old.addr_id;
update pkbc_address_history
set tbl_last_dt=current_timestamp()
where addr_id=old.addr_id;
end$$
delimiter ;

drop table if exists pkbc_customer_history;
create table pkbc_customer_history as select * from pkbc_customer where 1 = 2;
alter table pkbc_customer_history add constraint pk_customer_history primary key (cust_id);
```

```
delimiter $$
drop trigger if exists td_pkbc_customer;
create trigger td_pkbc_customer
before delete on pkbc_customer for each row
begin insert into pkbc_customer_history
select * from pkbc_customer
where cust_id = old.cust_id;
update pkbc_customer_history
set tbl_last_dt=current_timestamp()
where cust_id=old.cust_id;
end$$
delimiter ;

drop table if exists pkbc_ord_prod_history;
create table pkbc_ord_prod_history as select * from pkbc_ord_prod where 1 = 2;
alter table pkbc_ord_prod_history add constraint pk_ord_prod_history primary key (ord_prod_id);

delimiter $$
drop trigger if exists td_pkbc_ord_prod;
create trigger td_pkbc_ord_prod
before delete on pkbc_ord_prod for each row
begin insert into pkbc_ord_prod_history
select * from pkbc_ord_prod
where ord_prod_id = old.ord_prod_id;
update pkbc_ord_prod_history
set tbl_last_dt=current_timestamp()
where ord_prod_id=old.ord_prod_id;
end$$
delimiter ;
```

```
drop table if exists pkbc_orders_history;
create table pkbc_orders_history as select * from pkbc_orders where 1 = 2;
alter table pkbc_orders_history add constraint pk_orders_history primary key (order_id);

delimiter $$
drop trigger if exists td_pkbc_orders;
create trigger td_pkbc_orders
before delete on pkbc_orders for each row
begin insert into pkbc_orders_history
select * from pkbc_orders
where order_id = old.order_id;
update pkbc_orders_history
set tbl_last_dt=current_timestamp()
where order_id = old.order_id;
end$$
delimiter ;

drop table if exists pkbc_product_history;
create table pkbc_product_history as select * from pkbc_product where 1 = 2;
alter table pkbc_product_history add constraint pk_product_history primary key (product_id);

delimiter $$
drop trigger if exists td_pkbc_product;
create trigger td_pkbc_product
before delete on pkbc_product for each row
begin insert into pkbc_product_history
select * from pkbc_product
where product_id = old.product_id;
update pkbc_product_history
set tbl_last_dt=current_timestamp()
```

```
where product_id = old.product_id;
end$$
delimiter ;
```

STORED PROCEDURES

```
drop procedure if exists USP_UpsertCustomer;
delimiter $$
create procedure USP_UpsertCustomer
(
      in cust_name VARCHAR(100),
    in segment int,
    in email varchar(30),
    in `password` varchar(100)
)
begin
declare cust_id VARCHAR(20);
set cust_id = left(uuid(), 20);
insert into pkbc_customer (cust_id, cust_name, segment, email, `password`)
values (cust_id, cust_name, segment, email, `password`);
select cust_id;
end$$

delimiter ;

drop procedure if exists USP_GetCustomerById;
delimiter $$
create procedure USP_GetCustomerById(
      in cust_id VARCHAR(20)
```

```
)
begin
select * from pkbc_customer a where a.cust_id = cust_id limit 1;
end$$

delimiter ;

drop procedure if exists USP_GetCustomerByEmail;
delimiter $$
create procedure USP_GetCustomerByEmail(
      in email VARCHAR(30)
)
begin
select * from pkbc_customer a where a.email = email limit 1;
end$$

delimiter ;

drop procedure if exists USP_GetAllRegions;
delimiter $$
create procedure USP_GetAllRegions()
begin
select region_id, region_name from pkbc_region;
end$$

delimiter ;

drop procedure if exists USP_GetAllCountriesByRegion;
delimiter $$
create procedure USP_GetAllCountriesByRegion(
```

```
      in region_id int
)
begin
select c.country_id, c.country_name from pkbc_country c where c.region_id = region_id;
end$$

delimiter ;

drop procedure if exists USP_GetAllStatesByCountry;
delimiter $$
create procedure USP_GetAllStatesByCountry(
      in country_id int
)
begin
select s.state_id, s.state_name from pkbc_state s where s.country_id = country_id;
end$$

delimiter ;

drop procedure if exists USP_GetAllCitiesByState;
delimiter $$
create procedure USP_GetAllCitiesByState(
      in state_id int
)
begin
select c.city_id, c.city_name from pkbc_city c where c.state_id = state_id;
end$$

delimiter ;
```

```
drop procedure if exists USP_GetAllRegions;
delimiter $$
create procedure USP_GetAllRegions()
begin
select region_id, region_name from pkbc_region;
end$$

delimiter ;

drop procedure if exists USP_UpsertAddress;
delimiter $$
create procedure USP_UpsertAddress(
      in city_id int,
    in postal_code VARCHAR(10),
    in cust_id VARCHAR(20)
)
begin
insert into pkbc_address (city_id, postal_code, cust_id)
values (city_id, postal_code, cust_id);
select addr_id, city_id, postal_code, cust_id from pkbc_address where addr_id =
last_insert_id();
end$$

delimiter ;

drop procedure if exists USP_GetAddressByCustomer;
delimiter $$
create procedure USP_GetAddressByCustomer(
      in cust_id VARCHAR(20)
)
```

```
begin
select a.addr_id, c.city_name, s.state_name, co.country_name, r.region_name, a.postal_code
from pkbc_address a
left join pkbc_city c on a.city_id = c.city_id
left join pkbc_state s on s.state_id = c.state_id
left join pkbc_country co on co.country_id = s.country_id
left join pkbc_region r on co.region_id = r.region_id
where a.cust_id = cust_id;
end$$

delimiter ;

drop procedure if exists USP_UpdateAddress;
delimiter $$
create procedure USP_UpdateAddress(
      in addr_id int,
      in city_id int,
    in postal_code VARCHAR(10)
)
begin
update pkbc_address a
set
a.city_id = city_id,
a.postal_code = postal_code
where a.addr_id = addr_id;
select a.addr_id, a.city_id, a.postal_code, a.cust_id from pkbc_address a where a.addr_id =
addr_id;
end$$

delimiter ;
```

```
drop procedure if exists USP_UpdateCustomer;
delimiter $$
create procedure USP_UpdateCustomer(
      in cust_id VARCHAR(20),
    in cust_name VARCHAR(100),
    in segment int,
    in email varchar(30),
    in `password` varchar(100)
)
begin
update pkbc_customer c
set
c.cust_name = cust_name,
c.segment = segment,
c.email = email,
c.`password` = `password`
where c.cust_id = cust_id;
select c.cust_id, c.cust_name, c.segment, c.email from pkbc_customer c where c.cust_id =
cust_id;
end$$

delimiter ;

drop procedure if exists USP_GetAddressById;
delimiter $$
create procedure USP_GetAddressById(
      in addr_id int
)
begin
```

```
select a.city_id, s.state_id, co.country_id, r.region_id, a.postal_code, a.cust_id from
pkbc_address a
left join pkbc_city c on a.city_id = c.city_id
left join pkbc_state s on c.state_id = s.state_id
left join pkbc_country co on co.country_id = s.country_id
left join pkbc_region r on r.region_id = co.region_id
where a.addr_id = addr_id;
end$$

delimiter ;

drop procedure if exists USP_SetAndGetOTPByEmail;
delimiter $$
create procedure USP_SetAndGetOTPByEmail(
      in email VARCHAR(30)
)
begin
update pkbc_customer c
set c.otp_code = SUBSTRING(MD5(RAND()), 1, 5)
where c.email = email;
select c.otp_code from pkbc_customer c where c.email = email;
end$$

delimiter ;

drop procedure if exists USP_GetCustIdByEmailAndOTP;
delimiter $$
create procedure USP_GetCustIdByEmailAndOTP(
      in email VARCHAR(30),
    in otp VARCHAR(5)
```

```
)
begin
select c.cust_id from pkbc_customer c
where c.email = email and c.otp = otp;
end$$

delimiter ;

drop procedure if exists USP_GetAllCategory;
delimiter $$
create procedure USP_GetAllCategory()
begin
select category_id, category_name from pkbc_category;
end$$

delimiter ;

drop procedure if exists USP_GetAllSubcategoryByCategory;
delimiter $$
create procedure USP_GetAllSubcategoryByCategory(
      in category_id int
)
begin
select s.sub_category_id, s.sub_category_name from pkbc_sub_category s where s.category_id =
category_id;
end$$

delimiter ;

drop procedure if exists USP_UpsertProduct;
```

```
delimiter $$
create procedure USP_UpsertProduct(
      in product_name VARCHAR(200),
    in unit_price DECIMAL(10,2),
    in market int,
    in sub_category_id int
)
begin
declare sub_category_name VARCHAR(30);
declare category_name VARCHAR(30);
declare inserted_product_id VARCHAR(20);
select s.sub_category_name into sub_category_name from pkbc_sub_category s where
s.sub_category_id = sub_category_id limit 1;
select c.category_name into category_name from pkbc_sub_category s inner join
pkbc_category c on c.category_id = s.category_id limit 1;
select CONCAT(UPPER(SUBSTRING(category_name, 1, 3)), "-", UPPER(SUBSTRING(sub_category_name, 1,
2)), "-", SUBSTRING(UUID(), 1, 4)) into inserted_product_id;
insert into pkbc_product(
        product_id   ,
        unit_price     ,
        product_name ,
        market       ,
        sub_category_id
)
values (
      inserted_product_id,
    unit_price,
    product_name,
    market,
    sub_category_id
```

```
);
select * from pkbc_product where product_id = inserted_product_id limit 1;
end$$

delimiter ;

DROP PROCEDURE IF EXISTS USP_InsertOrdersData;

DELIMITER $$
CREATE PROCEDURE USP_InsertOrdersData
(
    OrderProdJSON JSON,
    in_addr_id bigint,
     in_cust_id varchar(20),
    in_ship_mode tinyint,
    in_ship_date datetime
)
BEGIN
     declare o_id VARCHAR(40);

    DECLARE jsonItemsLength BIGINT UNSIGNED DEFAULT JSON_LENGTH(`OrderProdJSON`);
     DECLARE idx BIGINT UNSIGNED DEFAULT 0;

     DECLARE EXIT HANDLER FOR SQLEXCEPTION
          BEGIN
            GET DIAGNOSTICS CONDITION 1 @sqlstate = RETURNED_SQLSTATE,
                                                    @errno = MYSQL_ERRNO, @text = MESSAGE_TEXT;
            SET @text = LEFT(@text, 100);
            SET @Full_Error = CONCAT("ERROR ", @errno, " (", @sqlstate, "): ", @text);
            ROLLBACK;
```

```
        SIGNAL SQLSTATE "45001" SET MESSAGE_TEXT = @Full_Error;
      END;

SET autocommit = 0;
 START TRANSACTION;

 set o_id = left(uuid(), 40);

 insert into pkbc_orders(
    order_id      ,
    order_date  ,
    is_returned
 ) values (
       o_id,
    NOW(),
    false
);

 DROP TEMPORARY TABLE IF EXISTS `temp_pkbc_ord_prod`;

 CREATE TEMPORARY TABLE  IF NOT EXISTS  `temp_pkbc_ord_prod`(
   `quantity` decimal(12,2) NOT NULL COMMENT 'Product Quantity for the order',
   `discount` decimal(10,2) NOT NULL COMMENT 'Discount for product of  the order',
   `shipping_cost` decimal(10,2) NOT NULL COMMENT 'Shipping cost for product of  the order',
   `profit` decimal(10,2) NOT NULL COMMENT 'Profit for product of  the order',
   `sales` bigint NOT NULL COMMENT 'Sales for the product of the order',
   `order_id` varchar(40) NOT NULL COMMENT 'Order Id',
   `product_id` varchar(20) NOT NULL COMMENT 'Product Id',
   `market` tinyint NOT NULL,
   `addr_id` bigint NOT NULL,
```

```
        `cust_id` varchar(20) NOT NULL COMMENT 'Customer Id',
        `ship_mode` tinyint NOT NULL COMMENT 'shipping  mode. 1: First Class , 2 : Second Class ,
3 : Same Day, 4 : Standard Class',
        `ship_date` datetime NOT NULL COMMENT 'Shipping Date'
      );

      WHILE idx < jsonItemsLength
      DO
            insert into `temp_pkbc_ord_prod`(
                quantity        ,
                discount        ,
                shipping_cost   ,
                profit          ,
                sales           ,
                order_id        ,
                product_id      ,
                market          ,
                addr_id         ,
                cust_id             ,
                ship_mode       ,
                ship_date
           )  values (
                JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx , '].quantity')))
           ,
                JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx , '].discount')))
           ,
                JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx , '].shipping_cost')))
      ,
                JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx , '].profit')))
           ,
```

```
                JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx , '].sales')))
          ,
                o_id,
                JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx , '].product_id')))
          ,
                case when JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx ,
'].market'))) = 'USCA'  then 1
                     when JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx ,
'].market'))) = 'Asia Pacific'  then 2
                     when JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx ,
'].market'))) = 'Europe'  then 3
                     when JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx ,
'].market'))) = 'Africa'  then 4
                     when JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx ,
'].market'))) = 'LATAM'  then 5
                     else 0 end ,
                in_addr_id            ,
                in_cust_id            ,
                in_ship_mode          ,
                in_ship_date
       );



      SET idx = idx + 1;
    END WHILE;

   insert into pkbc_ord_prod(
          quantity        ,
          discount        ,
```

```
            shipping_cost   ,
            profit          ,
            sales           ,
            order_id        ,
            product_id      ,
            market          ,
            addr_id         ,
            cust_id             ,
            ship_mode       ,
            ship_date
        ) select
            quantity        ,
            discount        ,
            shipping_cost   ,
            profit          ,
            sales           ,
            order_id        ,
            product_id      ,
            market          ,
            addr_id         ,
            cust_id             ,
            ship_mode       ,
            ship_date
        from `temp_pkbc_ord_prod`;


        DROP TEMPORARY TABLE IF EXISTS `temp_pkbc_ord_prod`;
    COMMIT WORK;

END $$
```

```
DELIMITER ;

drop procedure if exists USP_GetOrdersByCustomer;

delimiter $$
create procedure USP_GetOrdersByCustomer(
      in cust_id VARCHAR(20),
    in is_returned CHAR(1)
)
begin
select op.order_id, o.order_date, count(op.order_id) total_items
from pkbc_ord_prod op
left join pkbc_orders o on o.order_id = op.order_id
where op.cust_id = cust_id
and o.is_returned = is_returned
group by op.order_id;
end$$
delimiter ;

drop procedure if exists USP_ReturnOrder;

delimiter $$
create procedure USP_ReturnOrder(
      in order_id VARCHAR(40)
)
begin
update pkbc_orders o
set o.is_returned = '1'
where o.order_id = order_id;
select * from pkbc_orders o where o.order_id = order_id;
```

```
end$$
delimiter ;

drop procedure if exists USP_GetOrderProd;

delimiter $$
create procedure USP_GetOrderProd(
      in order_id VARCHAR(40)
)
begin
select
op.quantity,
op.discount,
op.shipping_cost,
op.profit,
op.sales,
op.ship_date,
op.ship_mode,
op.product_id,
p.product_name
from pkbc_ord_prod op
left join pkbc_product p on p.product_id = op.product_id
where op.order_id = order_id;
end$$
delimiter ;
```

b. **OLTP DML code**

INITIAL DATA INSERTS

```
drop procedure if exists USP_UpsertCustomer;
delimiter $$
create procedure USP_UpsertCustomer
(
      in cust_name VARCHAR(100),
    in segment int,
    in email varchar(30),
    in `password` varchar(100)
)
begin
declare cust_id VARCHAR(20);
set cust_id = left(uuid(), 20);
insert into pkbc_customer (cust_id, cust_name, segment, email, `password`)
values (cust_id, cust_name, segment, email, `password`);
select cust_id;
end$$

delimiter ;

drop procedure if exists USP_GetCustomerById;
delimiter $$
create procedure USP_GetCustomerById(
      in cust_id VARCHAR(20)
)
begin
select * from pkbc_customer a where a.cust_id = cust_id limit 1;
end$$
```

```
delimiter ;

drop procedure if exists USP_GetCustomerByEmail;
delimiter $$
create procedure USP_GetCustomerByEmail(
      in email VARCHAR(30)
)
begin
select * from pkbc_customer a where a.email = email limit 1;
end$$

delimiter ;

drop procedure if exists USP_GetAllRegions;
delimiter $$
create procedure USP_GetAllRegions()
begin
select region_id, region_name from pkbc_region;
end$$

delimiter ;

drop procedure if exists USP_GetAllCountriesByRegion;
delimiter $$
create procedure USP_GetAllCountriesByRegion(
      in region_id int
)
begin
select c.country_id, c.country_name from pkbc_country c where c.region_id = region_id;
end$$
```

```
delimiter ;

drop procedure if exists USP_GetAllStatesByCountry;
delimiter $$
create procedure USP_GetAllStatesByCountry(
      in country_id int
)
begin
select s.state_id, s.state_name from pkbc_state s where s.country_id = country_id;
end$$

delimiter ;

drop procedure if exists USP_GetAllCitiesByState;
delimiter $$
create procedure USP_GetAllCitiesByState(
      in state_id int
)
begin
select c.city_id, c.city_name from pkbc_city c where c.state_id = state_id;
end$$

delimiter ;

drop procedure if exists USP_GetAllRegions;
delimiter $$
create procedure USP_GetAllRegions()
begin
select region_id, region_name from pkbc_region;
```

```
end$$

delimiter ;

drop procedure if exists USP_UpsertAddress;
delimiter $$
create procedure USP_UpsertAddress(
      in city_id int,
    in postal_code VARCHAR(10),
    in cust_id VARCHAR(20)
)
begin
insert into pkbc_address (city_id, postal_code, cust_id)
values (city_id, postal_code, cust_id);
select addr_id, city_id, postal_code, cust_id from pkbc_address where addr_id =
last_insert_id();
end$$

delimiter ;

drop procedure if exists USP_GetAddressByCustomer;
delimiter $$
create procedure USP_GetAddressByCustomer(
      in cust_id VARCHAR(20)
)
begin
select a.addr_id, c.city_name, s.state_name, co.country_name, r.region_name, a.postal_code
from pkbc_address a
left join pkbc_city c on a.city_id = c.city_id
left join pkbc_state s on s.state_id = c.state_id
```

```
left join pkbc_country co on co.country_id = s.country_id
left join pkbc_region r on co.region_id = r.region_id
where a.cust_id = cust_id;
end$$

delimiter ;

drop procedure if exists USP_UpdateAddress;
delimiter $$
create procedure USP_UpdateAddress(
      in addr_id int,
      in city_id int,
    in postal_code VARCHAR(10)
)
begin
update pkbc_address a
set
a.city_id = city_id,
a.postal_code = postal_code
where a.addr_id = addr_id;
select a.addr_id, a.city_id, a.postal_code, a.cust_id from pkbc_address a where a.addr_id =
addr_id;
end$$

delimiter ;

drop procedure if exists USP_UpdateCustomer;
delimiter $$
create procedure USP_UpdateCustomer(
      in cust_id VARCHAR(20),
```

```
    in cust_name VARCHAR(100),
    in segment int,
    in email varchar(30),
    in `password` varchar(100)
)
begin
update pkbc_customer c
set
c.cust_name = cust_name,
c.segment = segment,
c.email = email,
c.`password` = `password`
where c.cust_id = cust_id;
select c.cust_id, c.cust_name, c.segment, c.email from pkbc_customer c where c.cust_id =
cust_id;
end$$

delimiter ;

drop procedure if exists USP_GetAddressById;
delimiter $$
create procedure USP_GetAddressById(
     in addr_id int
)
begin
select a.city_id, s.state_id, co.country_id, r.region_id, a.postal_code, a.cust_id from
pkbc_address a
left join pkbc_city c on a.city_id = c.city_id
left join pkbc_state s on c.state_id = s.state_id
left join pkbc_country co on co.country_id = s.country_id
```

```
left join pkbc_region r on r.region_id = co.region_id
where a.addr_id = addr_id;
end$$

delimiter ;

drop procedure if exists USP_SetAndGetOTPByEmail;
delimiter $$
create procedure USP_SetAndGetOTPByEmail(
      in email VARCHAR(30)
)
begin
update pkbc_customer c
set c.otp_code = SUBSTRING(MD5(RAND()), 1, 5)
where c.email = email;
select c.otp_code from pkbc_customer c where c.email = email;
end$$

delimiter ;

drop procedure if exists USP_GetCustIdByEmailAndOTP;
delimiter $$
create procedure USP_GetCustIdByEmailAndOTP(
      in email VARCHAR(30),
    in otp VARCHAR(5)
)
begin
select c.cust_id from pkbc_customer c
where c.email = email and c.otp = otp;
end$$
```

```
delimiter ;

drop procedure if exists USP_GetAllCategory;
delimiter $$
create procedure USP_GetAllCategory()
begin
select category_id, category_name from pkbc_category;
end$$

delimiter ;

drop procedure if exists USP_GetAllSubcategoryByCategory;
delimiter $$
create procedure USP_GetAllSubcategoryByCategory(
      in category_id int
)
begin
select s.sub_category_id, s.sub_category_name from pkbc_sub_category s where s.category_id =
category_id;
end$$

delimiter ;

drop procedure if exists USP_UpsertProduct;
delimiter $$
create procedure USP_UpsertProduct(
      in product_name VARCHAR(200),
    in unit_price DECIMAL(10,2),
    in market int,
```

```
    in sub_category_id int
)
begin
declare sub_category_name VARCHAR(30);
declare category_name VARCHAR(30);
declare inserted_product_id VARCHAR(20);
select s.sub_category_name into sub_category_name from pkbc_sub_category s where
s.sub_category_id = sub_category_id limit 1;
select c.category_name into category_name from pkbc_sub_category s inner join
pkbc_category c on c.category_id = s.category_id limit 1;
select CONCAT(UPPER(SUBSTRING(category_name, 1, 3)), "-", UPPER(SUBSTRING(sub_category_name, 1,
2)), "-", SUBSTRING(UUID(), 1, 4)) into inserted_product_id;
insert into pkbc_product(
        product_id   ,
        unit_price     ,
        product_name ,
        market       ,
        sub_category_id
)
values (
     inserted_product_id,
    unit_price,
    product_name,
    market,
    sub_category_id
);
select * from pkbc_product where product_id = inserted_product_id limit 1;
end$$

delimiter ;
```

```
DROP PROCEDURE IF EXISTS USP_InsertOrdersData;

DELIMITER $$
CREATE PROCEDURE USP_InsertOrdersData
(
    OrderProdJSON JSON,
    in_addr_id bigint,
     in_cust_id varchar(20),
    in_ship_mode tinyint,
    in_ship_date datetime
)
BEGIN
     declare o_id VARCHAR(40);

    DECLARE jsonItemsLength BIGINT UNSIGNED DEFAULT JSON_LENGTH(`OrderProdJSON`);
     DECLARE idx BIGINT UNSIGNED DEFAULT 0;

     DECLARE EXIT HANDLER FOR SQLEXCEPTION
          BEGIN
            GET DIAGNOSTICS CONDITION 1 @sqlstate = RETURNED_SQLSTATE,
                                                    @errno = MYSQL_ERRNO, @text = MESSAGE_TEXT;
            SET @text = LEFT(@text, 100);
            SET @Full_Error = CONCAT("ERROR ", @errno, " (", @sqlstate, "): ", @text);
            ROLLBACK;
            SIGNAL SQLSTATE "45001" SET MESSAGE_TEXT = @Full_Error;
          END;

    SET autocommit = 0;
     START TRANSACTION;
```

```
    set o_id = left(uuid(), 40);

    insert into pkbc_orders(
       order_id      ,
       order_date  ,
       is_returned
    ) values (
          o_id,
       NOW(),
       false
   );

    DROP TEMPORARY TABLE IF EXISTS `temp_pkbc_ord_prod`;

    CREATE TEMPORARY TABLE  IF NOT EXISTS  `temp_pkbc_ord_prod`(
       `quantity` decimal(12,2) NOT NULL COMMENT 'Product Quantity for the order',
       `discount` decimal(10,2) NOT NULL COMMENT 'Discount for product of  the order',
       `shipping_cost` decimal(10,2) NOT NULL COMMENT 'Shipping cost for product of  the order',
       `profit` decimal(10,2) NOT NULL COMMENT 'Profit for product of  the order',
       `sales` bigint NOT NULL COMMENT 'Sales for the product of the order',
       `order_id` varchar(40) NOT NULL COMMENT 'Order Id',
       `product_id` varchar(20) NOT NULL COMMENT 'Product Id',
       `market` tinyint NOT NULL,
       `addr_id` bigint NOT NULL,
       `cust_id` varchar(20) NOT NULL COMMENT 'Customer Id',
       `ship_mode` tinyint NOT NULL COMMENT 'shipping  mode. 1: First Class , 2 : Second Class ,
3 : Same Day, 4 : Standard Class',
       `ship_date` datetime NOT NULL COMMENT 'Shipping Date'
    );
```

```
WHILE idx < jsonItemsLength
DO
       insert into `temp_pkbc_ord_prod`(
            quantity        ,
            discount        ,
            shipping_cost   ,
            profit          ,
            sales           ,
            order_id        ,
            product_id      ,
            market          ,
            addr_id         ,
            cust_id             ,
            ship_mode       ,
            ship_date
      )  values (
            JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx , '].quantity')))
      ,
            JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx , '].discount')))
      ,
            JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx , '].shipping_cost')))
,
            JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx , '].profit')))
      ,
            JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx , '].sales')))
      ,
            o_id,
            JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx , '].product_id')))
      ,
```

```
                case when JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx ,
'].market'))) = 'USCA'  then 1
                     when JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx ,
'].market'))) = 'Asia Pacific'  then 2
                     when JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx ,
'].market'))) = 'Europe'  then 3
                     when JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx ,
'].market'))) = 'Africa'  then 4
                     when JSON_UNQUOTE(JSON_EXTRACT(OrderProdJSON, CONCAT('$[', idx ,
'].market'))) = 'LATAM'  then 5
                     else 0 end ,
                in_addr_id            ,
                in_cust_id            ,
                in_ship_mode          ,
                in_ship_date
        );



        SET idx = idx + 1;
      END WHILE;

    insert into pkbc_ord_prod(
           quantity        ,
           discount        ,
           shipping_cost   ,
           profit          ,
           sales           ,
           order_id        ,
           product_id      ,
```

```
            market          ,
            addr_id         ,
            cust_id             ,
            ship_mode       ,
            ship_date
        ) select
            quantity        ,
            discount        ,
            shipping_cost   ,
            profit          ,
            sales           ,
            order_id        ,
            product_id      ,
            market          ,
            addr_id         ,
            cust_id             ,
            ship_mode       ,
            ship_date
        from `temp_pkbc_ord_prod`;


        DROP TEMPORARY TABLE IF EXISTS `temp_pkbc_ord_prod`;
    COMMIT WORK;

END $$
DELIMITER ;

drop procedure if exists USP_GetOrdersByCustomer;

delimiter $$
```

```
create procedure USP_GetOrdersByCustomer(
      in cust_id VARCHAR(20),
    in is_returned CHAR(1)
)
begin
select op.order_id, o.order_date, count(op.order_id) total_items
from pkbc_ord_prod op
left join pkbc_orders o on o.order_id = op.order_id
where op.cust_id = cust_id
and o.is_returned = is_returned
group by op.order_id;
end$$
delimiter ;

drop procedure if exists USP_ReturnOrder;

delimiter $$
create procedure USP_ReturnOrder(
      in order_id VARCHAR(40)
)
begin
update pkbc_orders o
set o.is_returned = '1'
where o.order_id = order_id;
select * from pkbc_orders o where o.order_id = order_id;
end$$
delimiter ;

drop procedure if exists USP_GetOrderProd;
```

```
delimiter $$
create procedure USP_GetOrderProd(
      in order_id VARCHAR(40)
)
begin
select
op.quantity,
op.discount,
op.shipping_cost,
op.profit,
op.sales,
op.ship_date,
op.ship_mode,
op.product_id,
p.product_name
from pkbc_ord_prod op
left join pkbc_product p on p.product_id = op.product_id
where op.order_id = order_id;
end$$
delimiter ;
```

c. **OLTP Data Dictionary query and results (List of tables, constraint, columns and comments of each table)**

LIST OF TABLES

```
select table_name as 'Table Name'
from information_schema.tables
```

```
where table_schema = 'awesome_inc';
```

Table Name
pkbc_address
pkbc_address_history
pkbc_awesome_inc_orders
pkbc_awesome_inc_returns
pkbc_category
pkbc_city
pkbc_country
pkbc_customer
pkbc_customer_history
pkbc_ord_prod
pkbc_ord_prod_history
pkbc_orders
pkbc_orders_history
pkbc_product
pkbc_product_history
pkbc_region
pkbc_state
pkbc_sub_category

CONSTRAINTS

```
SELECT TABLE_NAME, COLUMN_NAME, CONSTRAINT_NAME, REFERENCED_TABLE_NAME, REFERENCED_COLUMN_NAME
FROM information_schema.KEY_COLUMN_USAGE
WHERE TABLE_SCHEMA = 'awesome_inc';
```

TABLE_NAME	COLUMN_NAME	CONSTRAINT_NAME	REFERENCED_TABLE_NAME	REFERENCED_COLUMN_NAME
pkbc_address	addr_id	PRIMARY	NULL	NULL
pkbc_address	city_id	pkbc_address_pkbc_city_fk	pkbc_city	city_id
pkbc_address	cust_id	pkbc_address_pkbc_customer_fk	pkbc_customer	cust_id

pkbc_address_history	addr_id	PRIMARY	NULL	NULL
pkbc_category	category_id	PRIMARY	NULL	NULL
pkbc_city	city_id	PRIMARY	NULL	NULL
pkbc_city	state_id	pkbc_city_pkbc_state_fk	pkbc_state	state_id
pkbc_country	country_id	PRIMARY	NULL	NULL
pkbc_country	region_id	pkbc_country_pkbc_region_fk	pkbc_region	region_id
pkbc_customer	cust_id	PRIMARY	NULL	NULL
pkbc_customer_history	cust_id	PRIMARY	NULL	NULL
pkbc_ord_prod	ord_prod_id	PRIMARY	NULL	NULL
pkbc_ord_prod	addr_id	pkbc_ord_prod_pkbc_address_fk	pkbc_address	addr_id
pkbc_ord_prod	cust_id	pkbc_ord_prod_pkbc_customer_fk	pkbc_customer	cust_id
pkbc_ord_prod	order_id	pkbc_ord_prod_pkbc_orders_fk	pkbc_orders	order_id
pkbc_ord_prod	product_id	pkbc_ord_prod_pkbc_product_fk	pkbc_product	product_id
pkbc_ord_prod	market	pkbc_ord_prod_pkbc_product_fk	pkbc_product	market
pkbc_ord_prod_history	ord_prod_id	PRIMARY	NULL	NULL
pkbc_orders	order_id	PRIMARY	NULL	NULL
pkbc_orders_history	order_id	PRIMARY	NULL	NULL
pkbc_product	product_id	PRIMARY	NULL	NULL
pkbc_product	market	PRIMARY	NULL	NULL
pkbc_product	sub_category_id	pkbc_sub_category_fk	pkbc_sub_category	sub_category_id
pkbc_product_history	product_id	PRIMARY	NULL	NULL
pkbc_region	region_id	PRIMARY	NULL	NULL
pkbc_state	state_id	PRIMARY	NULL	NULL
pkbc_state	country_id	pkbc_state_pkbc_country_fk	pkbc_country	country_id
pkbc_sub_category	sub_category_id	PRIMARY	NULL	NULL
pkbc_sub_category	category_id	pkbc_category_fk	pkbc_category	category_id

COLUMNS AND COMMENTS FOR EACH TABLE

```
select table_name, column_name, column_comment
from information_schema.columns
where table_schema  = 'awesome_inc';
```

TABLE_NAME	COLUMN_NAME	COLUMN_COMMENT
pkbc_address	addr_id	Address ID
pkbc_address	city_id	City id
pkbc_address	postal_code	Postal code
pkbc_address	cust_id	Customer Id
pkbc_address	tbl_last_dt	Timestamp for the row data added
pkbc_address_history	addr_id	Address ID
pkbc_address_history	city_id	City id
pkbc_address_history	postal_code	Postal code
pkbc_address_history	cust_id	Customer Id
pkbc_address_history	tbl_last_dt	Timestamp for the row data added
pkbc_awesome_inc_orders	Row ID	
pkbc_awesome_inc_orders	Order ID	
pkbc_awesome_inc_orders	Order Date	
pkbc_awesome_inc_orders	Ship Date	

pkbc_awesome_inc_orders	Ship Mode	
pkbc_awesome_inc_orders	Customer ID	
pkbc_awesome_inc_orders	Customer Name	
pkbc_awesome_inc_orders	Segment	
pkbc_awesome_inc_orders	Postal Code	
pkbc_awesome_inc_orders	City	
pkbc_awesome_inc_orders	State	
pkbc_awesome_inc_orders	Country	
pkbc_awesome_inc_orders	Region	
pkbc_awesome_inc_orders	Market	
pkbc_awesome_inc_orders	Product ID	
pkbc_awesome_inc_orders	Category	
pkbc_awesome_inc_orders	Sub-Category	
pkbc_awesome_inc_orders	Product Name	
pkbc_awesome_inc_orders	Sales	
pkbc_awesome_inc_ord	Quantity	

ers		
pkbc_awesome_inc_orders	Discount	
pkbc_awesome_inc_orders	Profit	
pkbc_awesome_inc_orders	Shipping Cost	
pkbc_awesome_inc_orders	Order Priority	
pkbc_awesome_inc_returns	Returned	
pkbc_awesome_inc_returns	Order ID	
pkbc_awesome_inc_returns	Region	
pkbc_category	category_id	Category id.
pkbc_category	category_name	Product Category Name
pkbc_category	tbl_last_dt	Timestamp for the row data added
pkbc_city	city_id	City Id
pkbc_city	city_name	City Name
pkbc_city	state_id	State id
pkbc_city	tbl_last_dt	Timestamp for the row data added
pkbc_country	country_id	Country id
pkbc_country	country_name	Country Name
pkbc_country	region_id	Region id
pkbc_country	tbl_last_dt	Timestamp for the row data added
pkbc_customer	cust_id	Customer Id.
pkbc_customer	cust_name	Customer name
pkbc_customer	segment	Customer Segment. 1: Consumer , 2: Corporate, 3: Home Office

pkbc_customer	email	Email of the customer
pkbc_customer	password	Password of the login of the customer
pkbc_customer	tbl_last_dt	Timestamp for the row data added
pkbc_customer	otp_code	
pkbc_customer_history	cust_id	Customer Id.
pkbc_customer_history	cust_name	Customer name
pkbc_customer_history	segment	Customer Segment. 1: Consumer , 2: Corporate, 3: Home Office
pkbc_customer_history	email	Email of the customer
pkbc_customer_history	password	Password of the login of the customer
pkbc_customer_history	tbl_last_dt	Timestamp for the row data added
pkbc_customer_history	otp_code	
pkbc_ord_prod	ord_prod_id	unique id for all orders and products
pkbc_ord_prod	quantity	Product Quantity for the order
pkbc_ord_prod	discount	Discount for product of the order
pkbc_ord_prod	shipping_cost	Shipping cost for product of the order
pkbc_ord_prod	profit	Profit for product of the order
pkbc_ord_prod	sales	Sales for the product of the order
pkbc_ord_prod	order_id	Order Id
pkbc_ord_prod	product_id	Product Id
pkbc_ord_prod	tbl_last_dt	Timestamp for the row data added
pkbc_ord_prod	market	
pkbc_ord_prod	addr_id	
pkbc_ord_prod	cust_id	Customer Id
pkbc_ord_prod	ship_mode	shipping mode. 1: First Class , 2 : Second Class , 3 : Same Day, 4 : Standard Class
pkbc_ord_prod	ship_date	Shipping Date
pkbc_ord_prod_history	ord_prod_id	unique id for all orders and products
pkbc_ord_prod_history	quantity	Product Quantity for the order

pkbc_ord_prod_history	discount	Discount for product of the order
pkbc_ord_prod_history	shipping_cost	Shipping cost for product of the order
pkbc_ord_prod_history	profit	Profit for product of the order
pkbc_ord_prod_history	sales	Sales for the product of the order
pkbc_ord_prod_history	order_id	Order Id
pkbc_ord_prod_history	product_id	Product Id
pkbc_ord_prod_history	tbl_last_dt	Timestamp for the row data added
pkbc_ord_prod_history	market	
pkbc_ord_prod_history	addr_id	
pkbc_ord_prod_history	cust_id	Customer Id
pkbc_ord_prod_history	ship_mode	shipping mode. 1: First Class , 2 : Second Class , 3 : Same Day, 4 : Standard Class
pkbc_ord_prod_history	ship_date	Shipping Date
pkbc_orders	order_id	Order Id
pkbc_orders	order_date	Order Date
pkbc_orders	is_returned	Is order returned. True : returned, False: not returned
pkbc_orders	tbl_last_dt	Timestamp for the row data added
pkbc_orders_history	order_id	Order Id
pkbc_orders_history	order_date	Order Date
pkbc_orders_history	is_returned	Is order returned. True : returned, False: not returned
pkbc_orders_history	tbl_last_dt	Timestamp for the row data added
pkbc_product	product_id	Product id.
pkbc_product	product_name	Product Name
pkbc_product	market	Product Market. 1: Africa, 2: Asia Pacific, 3: Europe, 4: LATAM, 5: USCA
pkbc_product	sub_category_id	Category Id
pkbc_product	tbl_last_dt	Timestamp for the row data added
pkbc_product	unit_price	
pkbc_product_history	product_id	Product id.

pkbc_product_history	product_name	Product Name
pkbc_product_history	market	Product Market. 1: Africa, 2: Asia Pacific, 3: Europe, 4: LATAM, 5: USCA
pkbc_product_history	sub_category_id	Category Id
pkbc_product_history	tbl_last_dt	Timestamp for the row data added
pkbc_product_history	unit_price	
pkbc_region	region_id	Country id
pkbc_region	region_name	Region Name
pkbc_region	tbl_last_dt	Timestamp for the row data added
pkbc_state	state_id	state id
pkbc_state	state_name	State Name
pkbc_state	country_id	Country id
pkbc_state	tbl_last_dt	Timestamp for the row data added
pkbc_sub_category	sub_category_id	Sub Category id.
pkbc_sub_category	sub_category_name	Product Sub Category Name
pkbc_sub_category	category_id	Category id.
pkbc_sub_category	tbl_last_dt	Timestamp for the row data added

INDEXES

```
SELECT TABLE_NAME, INDEX_NAME, COLUMN_NAME, SEQ_IN_INDEX
FROM information_schema.STATISTICS
WHERE TABLE_SCHEMA = 'awesome_inc';
```

TABLE_NAME	INDEX_NAME	COLUMN_NAME	SEQ_IN_INDEX

pkbc_category	PRIMARY	category_id	1
pkbc_orders	PRIMARY	order_id	1
pkbc_region	PRIMARY	region_id	1
pkbc_address	PRIMARY	addr_id	1
pkbc_address	cust_id_idx	cust_id	1
pkbc_address	pkbc_address_pkbc_city_fk	city_id	1
pkbc_country	PRIMARY	country_id	1
pkbc_country	pkbc_country_pkbc_region_fk	region_id	1
pkbc_state	PRIMARY	state_id	1
pkbc_state	pkbc_state_pkbc_country_fk	country_id	1
pkbc_city	PRIMARY	city_id	1
pkbc_city	pkbc_city_pkbc_state_fk	state_id	1
pkbc_ord_prod	PRIMARY	ord_prod_id	1
pkbc_ord_prod	cust_id_idx	cust_id	1
pkbc_ord_prod	product_id_idx	product_id	1
pkbc_ord_prod	pkbc_ord_prod_pkbc_address_fk	addr_id	1
pkbc_ord_prod	pkbc_ord_prod_pkbc_orders_fk	order_id	1
pkbc_ord_prod	pkbc_ord_prod_pkbc_product_fk	product_id	1
pkbc_ord_prod	pkbc_ord_prod_pkbc_product_fk	market	2
pkbc_sub_category	PRIMARY	sub_category_id	1
pkbc_sub_category	pkbc_category_fk	category_id	1
pkbc_customer	PRIMARY	cust_id	1
pkbc_product	PRIMARY	product_id	1
pkbc_product	PRIMARY	market	2
pkbc_product	product_id_idx	product_id	1
pkbc_product	pkbc_sub_category_fk	sub_category_id	1
pkbc_address_history	PRIMARY	addr_id	1

pkbc_customer_history	PRIMARY	cust_id	1
pkbc_ord_prod_history	PRIMARY	ord_prod_id	1
pkbc_orders_history	PRIMARY	order_id	1
pkbc_product_history	PRIMARY	product_id	1

d. **DW DDL code (Tables, Constraints, Trigger, Partitioned tables, any function/ procedure created etc.)**

//DW Tables and Constraints

```
CREATE TABLE dim_pkbc_address (
   address_id   NUMBER(10) NOT NULL,
   postal_code  VARCHAR2(10),
   city_name    VARCHAR2(50) NOT NULL,
   state_name   VARCHAR2(50) NOT NULL,
   country_name VARCHAR2(50) NOT NULL,
   region_name  VARCHAR2(50) NOT NULL
);

COMMENT ON COLUMN dim_pkbc_address.address_id IS
   'Location ID';

COMMENT ON COLUMN dim_pkbc_address.city_name IS
   'City Name';

COMMENT ON COLUMN dim_pkbc_address.state_name IS
   'State name';

COMMENT ON COLUMN dim_pkbc_address.country_name IS
   'Country name';
```

```
COMMENT ON COLUMN dim_pkbc_address.region_name IS
   'Region name';

ALTER TABLE dim_pkbc_address ADD CONSTRAINT dim_pkbc_address_pk PRIMARY KEY ( address_id );

CREATE TABLE dim_pkbc_customer (
   cust_id   VARCHAR2(20) NOT NULL,
   cust_name VARCHAR2(50) NOT NULL,
   email     VARCHAR2(30) NOT NULL,
         password  VARCHAR2(100) NOT NULL ,
   segment   NUMBER(2) NOT NULL
);

COMMENT ON COLUMN dim_pkbc_customer.password IS 'Password of the login of the customer';

COMMENT ON COLUMN dim_pkbc_customer.cust_id IS
   'Customer Id.';

COMMENT ON COLUMN dim_pkbc_customer.cust_name IS
   'Customer name';

COMMENT ON COLUMN dim_pkbc_customer.segment IS
   'Customer Segment. 1: Consumer , 2: Corporate, 3: Home Office';

COMMENT ON COLUMN dim_pkbc_customer.email IS
   'Email of the customer';

ALTER TABLE dim_pkbc_customer ADD CONSTRAINT dim_pkbc_customer_pk PRIMARY KEY ( cust_id );

CREATE TABLE dim_pkbc_date AS
SELECT
     n AS ENTRY_ID,
     TO_DATE('31/12/1970','DD/MM/YYYY') + NUMTODSINTERVAL(n,'day') AS Full_Date,
```

```
    TO_CHAR(TO_DATE('31/12/1970','DD/MM/YYYY') + NUMTODSINTERVAL(n,'day'),'DD') AS Days,
    TO_CHAR(TO_DATE('31/12/1970','DD/MM/YYYY') + NUMTODSINTERVAL(n,'day'),'Mon') AS Month_Short,
    TO_CHAR(TO_DATE('31/12/1970','DD/MM/YYYY') + NUMTODSINTERVAL(n,'day'),'MM') AS Month_Num,
    TO_CHAR(TO_DATE('31/12/1970','DD/MM/YYYY') + NUMTODSINTERVAL(n,'day'),'Month') AS Month_Long,
    TO_CHAR(TO_DATE('31/12/1970','DD/MM/YYYY') + NUMTODSINTERVAL(n,'day'),'D') AS Day_Of_Week_Short,
    TO_CHAR(TO_DATE('31/12/1970','DD/MM/YYYY') + NUMTODSINTERVAL(n,'day'),'Day') AS Day_Of_Week_Long,
    TO_CHAR(TO_DATE('31/12/1970','DD/MM/YYYY') + NUMTODSINTERVAL(n,'day'),'YYYY') AS Year,
    CASE
      WHEN TO_CHAR(TO_DATE('31/12/1970','DD/MM/YYYY') + NUMTODSINTERVAL(n,'day'),'MM') IN (1,2,3) THEN 'Q1'
      WHEN TO_CHAR(TO_DATE('31/12/1970','DD/MM/YYYY') + NUMTODSINTERVAL(n,'day'),'MM') IN (4,5,6) THEN 'Q2'
      WHEN TO_CHAR(TO_DATE('31/12/1970','DD/MM/YYYY') + NUMTODSINTERVAL(n,'day'),'MM') IN (7,8,9) THEN 'Q3'
    ELSE
     'Q4'
    END Quarter,
    TO_CHAR(TO_DATE('31/12/1970','DD/MM/YYYY') + NUMTODSINTERVAL(n,'day'),'MM')||
    TO_CHAR(TO_DATE('31/12/1970','DD/MM/YYYY') + NUMTODSINTERVAL(n,'day'),'DD')||
    TO_CHAR(TO_DATE('31/12/1970','DD/MM/YYYY') + NUMTODSINTERVAL(n,'day'),'YYYY')
    AS DATE_ID
FROM (
  select level n
  from dual
  connect by level <= 36600);  -- this will create dates for 10  years starting 01/01/1971


COMMENT ON COLUMN dim_pkbc_date.month_short IS
   'Month';

COMMENT ON COLUMN dim_pkbc_date.month_long IS
   'Month';

COMMENT ON COLUMN dim_pkbc_date.year IS
   'Year';

ALTER TABLE dim_pkbc_date ADD CONSTRAINT dim_pkbc_date_pk PRIMARY KEY ( date_id );
```

```
CREATE TABLE dim_pkbc_product (
    product_id    VARCHAR2(20) NOT NULL,
    product_name VARCHAR2(200) NOT NULL,
    unit_price    NUMBER(10, 2) NOT NULL,
    market        NUMBER(1) NOT NULL
);

COMMENT ON COLUMN dim_pkbc_product.product_id IS
    'Product  id.';

COMMENT ON COLUMN dim_pkbc_product.product_name IS
    'Product Name';

COMMENT ON COLUMN dim_pkbc_product.unit_price IS
    'Price';

COMMENT ON COLUMN dim_pkbc_product.market IS
    'Product Market. 1: Africa, 2: Asia Pacific, 3: Europe, 4: LATAM, 5: USCA


';

ALTER TABLE dim_pkbc_product ADD CONSTRAINT dim_pkbc_product_pk PRIMARY KEY ( product_id, market );

CREATE TABLE dim_pkbc_sub_category (
    sub_category_id    NUMBER(5) NOT NULL,
    sub_category_name VARCHAR2(30) NOT NULL,
    category_name      VARCHAR2(30) NOT NULL
);

COMMENT ON COLUMN dim_pkbc_sub_category.sub_category_id IS
    'Category id.';
```

```
COMMENT ON COLUMN dim_pkbc_sub_category.sub_category_name IS
   'Product Category Name';

COMMENT ON COLUMN dim_pkbc_sub_category.category_name IS
   'Category name';

ALTER TABLE dim_pkbc_sub_category ADD CONSTRAINT dim_pkbc_sub_category_pk PRIMARY KEY ( sub_category_id
);
```

// Partition table

```
CREATE TABLE fact_pkbc_orders (
   row_id          NUMBER(12) NOT NULL,
   order_id        VARCHAR2(40) NOT NULL,
   order_date      DATE NOT NULL,
   is_returned     CHAR(1) NOT NULL,
   quantity        NUMBER(12, 2) NOT NULL,
   discount        NUMBER(10, 2) NOT NULL,
   shipping_cost   NUMBER(10, 2) NOT NULL,
   profit          NUMBER(10, 2) NOT NULL,
   sales           NUMBER(12) NOT NULL,
   ship_date       DATE NOT NULL,
   ship_mode       NUMBER(1) NOT NULL,
   address_id      NUMBER(10) NOT NULL,
   cust_id         VARCHAR2(20) NOT NULL,
   sub_category_id NUMBER(5) NOT NULL,
   product_id      VARCHAR2(20) NOT NULL,
         market       NUMBER(1) NOT NULL,
   date_id         VARCHAR2(8) NOT NULL
)PARTITION BY RANGE (order_date)
     ( PARTITION P1 VALUES less than(to_date('01-JAN-2012', 'DD-MON-YYYY')),
       PARTITION P2 VALUES less than(to_date('01-JAN-2013', 'DD-MON-YYYY')),
       PARTITION P3 VALUES less than(to_date('01-JAN-2014', 'DD-MON-YYYY')),
       PARTITION P4 VALUES less than(to_date('01-JAN-2015', 'DD-MON-YYYY')),
```

```
    PARTITION P5 VALUES less than(to_date('01-JAN-2016', 'DD-MON-YYYY')),
    PARTITION P6 VALUES less than(to_date('01-JAN-2017', 'DD-MON-YYYY')),
    PARTITION P7 VALUES less than(to_date('01-JAN-2018', 'DD-MON-YYYY')),
    PARTITION P8 VALUES less than(to_date('01-JAN-2019', 'DD-MON-YYYY')),
    PARTITION P9 VALUES less than(to_date('01-JAN-2020', 'DD-MON-YYYY')),
    PARTITION P10 VALUES less than(to_date('01-JAN-2021', 'DD-MON-YYYY')),
    PARTITION P11 VALUES less than(to_date('01-JAN-2022', 'DD-MON-YYYY')),
    PARTITION P12 VALUES less than(to_date('01-JAN-2023', 'DD-MON-YYYY')),
    PARTITION P13 VALUES less than(to_date('01-JAN-2024', 'DD-MON-YYYY'))
  );



COMMENT ON COLUMN fact_pkbc_orders.order_id IS
   'Order id';

COMMENT ON COLUMN fact_pkbc_orders.order_date IS
   'Oder Date';

COMMENT ON COLUMN fact_pkbc_orders.is_returned IS
   'Is returned';

COMMENT ON COLUMN fact_pkbc_orders.quantity IS
   'Product Quantity for the order';

COMMENT ON COLUMN fact_pkbc_orders.discount IS
   'Discount for product of  the order';

COMMENT ON COLUMN fact_pkbc_orders.shipping_cost IS
   'Shipping cost for product of  the order';

COMMENT ON COLUMN fact_pkbc_orders.profit IS
   'Profit for product of  the order';
```

```
COMMENT ON COLUMN fact_pkbc_orders.sales IS
   'Sales for the product of the order';


COMMENT ON COLUMN fact_pkbc_orders.ship_date IS
   'Shipping Date';

COMMENT ON COLUMN fact_pkbc_orders.ship_mode IS
   'Shipping Mode';

ALTER TABLE fact_pkbc_orders ADD CONSTRAINT fact_pkbc_orders_pk PRIMARY KEY ( row_id );

ALTER TABLE fact_pkbc_orders
   ADD CONSTRAINT orders_address_fk FOREIGN KEY ( address_id )
      REFERENCES dim_pkbc_address ( address_id );

ALTER TABLE fact_pkbc_orders
   ADD CONSTRAINT orders_customer_fk FOREIGN KEY ( cust_id )
      REFERENCES dim_pkbc_customer ( cust_id );

ALTER TABLE fact_pkbc_orders
   ADD CONSTRAINT orders_date_fk FOREIGN KEY ( date_id )
      REFERENCES dim_pkbc_date ( date_id );

ALTER TABLE fact_pkbc_orders
   ADD CONSTRAINT orders_product_fk FOREIGN KEY ( product_id, market )
      REFERENCES dim_pkbc_product ( product_id, market );


ALTER TABLE fact_pkbc_orders
   ADD CONSTRAINT orders_sub_category_fk FOREIGN KEY ( sub_category_id )
      REFERENCES dim_pkbc_sub_category ( sub_category_id );

-- add tbl_last_date for all tables
```

```
alter table dim_pkbc_address add tbl_last_date timestamp default sysdate;
alter table dim_pkbc_customer add tbl_last_date timestamp default sysdate;
alter table dim_pkbc_date add tbl_last_date timestamp default sysdate;
alter table dim_pkbc_product add tbl_last_date timestamp default sysdate;
alter table dim_pkbc_sub_category add tbl_last_date timestamp default sysdate;
alter table fact_pkbc_orders add tbl_last_date timestamp default sysdate;

COMMENT ON COLUMN dim_pkbc_address.tbl_last_date IS
   'Timestamp for the row data added';

COMMENT ON COLUMN dim_pkbc_customer.tbl_last_date IS
   'Timestamp for the row data added';

COMMENT ON COLUMN dim_pkbc_date.tbl_last_date IS
   'Timestamp for the row data added';

COMMENT ON COLUMN dim_pkbc_product.tbl_last_date IS
   'Timestamp for the row data added';

COMMENT ON COLUMN dim_pkbc_sub_category.tbl_last_date IS
   'Timestamp for the row data added';

COMMENT ON COLUMN fact_pkbc_orders.tbl_last_date IS
   'Timestamp for the row data added';
```

Procedures and functions:

```
create or replace PROCEDURE  load_merge_dim_pkbc_address IS
err_code NUMBER;
err_msg VARCHAR2(32000);
```

```
count_n number;
BEGIN
   MERGE INTO dim_pkbc_address a
   USING stg_pkbc_dim_address b
   ON (a.address_id = b.address_id)
 WHEN MATCHED THEN
   UPDATE SET
          a.postal_code     = b.postal_code ,
          a.city_name       = b.city_name   ,
          a.state_name      = b.state_name  ,
          a.country_name    = b.country_name,
          a.region_name     = b.region_name ,
     a.TBL_LAST_DATE = to_timestamp(b.tbl_last_dt,'RRRR-MM-DD HH24:MI:SS')
 WHEN NOT MATCHED THEN
    INSERT (
          address_id    ,
          postal_code   ,
          city_name     ,
          state_name    ,
          country_name  ,
          region_name   ,
          TBL_LAST_DATE
          )
    VALUES (
          b.address_id ,
          b.postal_code ,
          b.city_name   ,
          b.state_name  ,
          b.country_name,
          b.region_name ,
          to_timestamp(b.tbl_last_dt,'RRRR-MM-DD HH24:MI:SS'));

    select count(*) into count_n from dim_pkbc_address;
    dbms_output.put_line('count '||  count_n);
```

```
    commit;
EXCEPTION
   WHEN OTHERS THEN
       err_code := SQLCODE;
       err_msg  := SQLERRM;
       dbms_output.put_line('Error code ' || err_code || ': ' || err_msg);

END;



create or replace PROCEDURE  load_merge_dim_pkbc_customer IS
err_code NUMBER;
err_msg VARCHAR2(32000);
count_n number;
BEGIN
   MERGE INTO dim_pkbc_customer a
   USING stg_pkbc_dim_customer b
   ON (a.cust_id = b.cust_id)
 WHEN MATCHED THEN
   UPDATE SET
          a.cust_name              = b.cust_name       ,
          a.email            = b.email             ,
          a.password       = b.password        ,
          a.segment        = b.segment         ,
    a.TBL_LAST_DATE = to_timestamp(b.tbl_last_dt,'RRRR-MM-DD HH24:MI:SS')
 WHEN NOT MATCHED THEN
    INSERT (
          cust_id                          ,
          cust_name       ,
          email           ,
          password        ,
          segment         ,
          TBL_LAST_DATE
```

```
        )
    VALUES (
        b.cust_id               ,
        b.cust_name             ,
        b.email         ,
        b.password      ,
        b.segment       ,
        to_timestamp(b.tbl_last_dt,'RRRR-MM-DD HH24:MI:SS'));

    select count(*) into count_n from dim_pkbc_customer;
    dbms_output.put_line('count '||  count_n);
    commit;
EXCEPTION
   WHEN OTHERS THEN
      err_code := SQLCODE;
      err_msg  := SQLERRM;
      dbms_output.put_line('Error code ' || err_code || ': ' || err_msg);

END;


create or replace PROCEDURE  load_merge_dim_pkbc_product IS
err_code NUMBER;
err_msg VARCHAR2(32000);
count_n number;
BEGIN
   MERGE INTO dim_pkbc_product a
   USING stg_pkbc_dim_product b
   ON (a.product_id = b.product_id and a.market = b.market)
 WHEN MATCHED THEN
   UPDATE SET
        a.product_name    = b.product_name      ,
        a.unit_price      = b.unit_price        ,
    a.TBL_LAST_DATE = to_timestamp(b.tbl_last_dt,'RRRR-MM-DD HH24:MI:SS')
```

```
  WHEN NOT MATCHED THEN
     INSERT (
           product_id     ,
           product_name     ,
           unit_price       ,
           market           ,
           TBL_LAST_DATE
           )
     VALUES (
           b.product_id                ,
           b.product_name              ,
           b.unit_price                ,
           b.market                    ,
       to_timestamp(b.tbl_last_dt,'RRRR-MM-DD HH24:MI:SS'));

     select count(*) into count_n from dim_pkbc_product;
     dbms_output.put_line('count '||  count_n);
     commit;
EXCEPTION
   WHEN OTHERS THEN
       err_code := SQLCODE;
       err_msg  := SQLERRM;
       dbms_output.put_line('Error code ' || err_code || ': ' || err_msg);

END;


create or replace PROCEDURE  load_merge_dim_pkbc_sub_category IS
err_code NUMBER;
err_msg VARCHAR2(32000);
count_n number;
BEGIN
   MERGE INTO dim_pkbc_sub_category a
   USING stg_pkbc_dim_sub_category b
```

```
   ON (a.sub_category_id = b.sub_category_id)
  WHEN MATCHED THEN
    UPDATE SET
           a.sub_category_name = b.sub_category_name     ,
           a.category_name     = b.category_name     ,
      a.TBL_LAST_DATE = to_timestamp(b.tbl_last_dt,'RRRR-MM-DD HH24:MI:SS')
  WHEN NOT MATCHED THEN
      INSERT (
           sub_category_id                ,
           sub_category_name     ,
           category_name         ,
           TBL_LAST_DATE
           )
      VALUES (
           b.sub_category_id    ,
           b.sub_category_name   ,
           b.category_name       ,
        to_timestamp(b.tbl_last_dt,'RRRR-MM-DD HH24:MI:SS'));

      select count(*) into count_n from dim_pkbc_sub_category;
      dbms_output.put_line('count '||  count_n);
      commit;
EXCEPTION
    WHEN OTHERS THEN
        err_code := SQLCODE;
        err_msg  := SQLERRM;
        dbms_output.put_line('Error code ' || err_code || ': ' || err_msg);

END;

create or replace PROCEDURE  load_merge_fact_pkbc_orders IS
err_code NUMBER;
err_msg VARCHAR2(32000);
count_n number;
```

```
BEGIN
   MERGE INTO fact_pkbc_orders a
   USING stg_pkbc_fact_orders b
   ON (a.row_id = b.row_id)
 WHEN MATCHED THEN
   UPDATE SET
      a.order_id        = b.order_id                ,
      a.order_date      = b.order_date      ,
      a.is_returned     = b.is_returned     ,
      a.quantity        = b.quantity        ,
      a.discount        = b.discount        ,
      a.shipping_cost   = b.shipping_cost   ,
      a.profit          = b.profit          ,
      a.sales           = b.sales           ,
      a.ship_date       = b.ship_date       ,
      a.ship_mode       = b.ship_mode       ,
      a.address_id      = b.address_id      ,
      a.cust_id         = b.cust_id         ,
      a.sub_category_id  = b.sub_category_id ,
      a.product_id      = b.product_id      ,
                  a.market              = b.market                    ,
      a.date_id         = b.date_id         ,
    a.TBL_LAST_DATE = to_timestamp(b.tbl_last_dt,'RRRR-MM-DD HH24:MI:SS')
 WHEN NOT MATCHED THEN
    INSERT (
          row_id                    ,
          order_id          ,
          order_date        ,
          is_returned       ,
          quantity          ,
          discount          ,
          shipping_cost     ,
          profit            ,
          sales             ,
```

```
            ship_date         ,
            ship_mode         ,
            address_id        ,
            cust_id           ,
            sub_category_id   ,
            product_id        ,
            market                          ,
            date_id           ,
            TBL_LAST_DATE
            )
        VALUES (
            b.row_id                        ,
            b.order_id        ,
            b.order_date      ,
            b.is_returned     ,
            b.quantity        ,
            b.discount        ,
            b.shipping_cost   ,
            b.profit          ,
            b.sales           ,
            b.ship_date       ,
            b.ship_mode       ,
            b.address_id      ,
            b.cust_id         ,
            b.sub_category_id ,
            b.product_id      ,
            b.market                        ,
            b.date_id         ,
        to_timestamp(b.tbl_last_dt,'RRRR-MM-DD HH24:MI:SS'));

        select count(*) into count_n from fact_pkbc_orders;
        dbms_output.put_line('count '||  count_n);
        commit;
    EXCEPTION
```

```
    WHEN OTHERS THEN
        err_code := SQLCODE;
        err_msg  := SQLERRM;
        dbms_output.put_line('Error code ' || err_code || ': ' || err_msg);

END;
```

e. ETL code used

Example for loading data in fact_pkbc_orders table:

Extract done in the procedure code of mysql:

```
use awesome_inc;

drop procedure if exists USP_Run_ETL_Full_Extract;
delimiter $$
create procedure USP_Run_ETL_Full_Extract()
BEGIN

--

select
  a.addr_id as address_id,
          a.postal_code as postal_code,
       b.city_name as city_name,
       c.state_name as state_name,
       d.country_name as country_name,
       e.region_name as region_name,
          a.tbl_last_dt
into outfile 'fullstg_pkbc_dim_address.csv'
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ""
  LINES TERMINATED BY '\n'
```

```
  FROM pkbc_address a
            INNER JOIN pkbc_city b on a.city_id = b.city_id
        INNER JOIN pkbc_state c on b.state_id = c.state_id
        INNER JOIN pkbc_country d on c.country_id = d.country_id
        INNER JOIN pkbc_region e on d.region_id = e.region_id;

--

select    cust_id ,
     cust_name,
     segment  ,
     email   ,
     `password` ,
   tbl_last_dt
into outfile 'fullstg_pkbc_dim_customer.csv'
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ""
  LINES TERMINATED BY '\n'
  FROM pkbc_customer ;

  --

  select
    product_id ,
    product_name,
    unit_price ,
    market ,
    tbl_last_dt
into outfile 'fullstg_pkbc_dim_product.csv'
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ""
  LINES TERMINATED BY '\n'
  FROM pkbc_product ;


--
```

```
select * from (
SELECT 'sub_category_id', 'sub_category_name', 'category_name','tbl_last_dt'
UNION ALL
 select
   sub_category_id ,
   sub_category_name,
   category_name ,
          a.tbl_last_dt
 FROM pkbc_sub_category a
          inner join pkbc_category b on a.category_id = b.category_id
) resulting_set
into outfile 'fullstg_pkbc_dim_sub_category.csv'
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ""
  LINES TERMINATED BY '\n'
 ;

--
select
 ord_prod_id as row_id,
 a.order_id,
 DATE_FORMAT(b.order_date, '%Y%m%d %H:%i'),
 b.is_returned,
 quantity,
 discount ,
 shipping_cost   ,
 profit     ,
 sales   ,
 DATE_FORMAT(ship_date, '%Y%m%d %H:%i'),
 ship_mode   ,
 addr_id as address_id    ,
 cust_id   ,
 sub_category_id ,
 a.product_id  ,
 a.market   ,
```

```
 DATE_FORMAT(order_date,'%m%d%Y'),
 a.tbl_last_dt
into outfile 'fullstg_pkbc_fact_orders.csv'
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ""
  LINES TERMINATED BY '\n'
  FROM pkbc_ord_prod a
  INNER JOIN pkbc_orders b on a.order_id = b.order_id
  INNER JOIN pkbc_product c on a.product_id = c.product_id and a.market = c.market;

  insert into etl_extract_date(last_extract_date) values ( now() );

END;

-- CALL USP_Run_ETL_Full_Extract();
```

Control file:

```
LOAD DATA
INTO TABLE stg_pkbc_fact_orders
TRUNCATE
FIELDS TERMINATED BY ',' optionally enclosed by ""
NULLIF = "NULL"
DATE FORMAT "yyyy-mm-dd HH24:MI"
  (

  row_id          ,
  order_id        ,
  order_date date "yyyy-mm-dd HH24:MI"    ,
  is_returned     ,
  quantity        ,
  discount        ,
```

```
    shipping_cost    ,
    profit           ,
    sales            ,
    ship_date  date "yyyy-mm-dd HH24:MI"      ,
    ship_mode        ,
    address_id       ,
    cust_id          ,
    sub_category_id  ,
    product_id       ,
    market                              ,
    date_id          ,
    TBL_LAST_DT
        )
```

SQL Loader command:

```
sqlldr CONTROL='/media/sf_VMSHARE/control-files/fullstg_pkbc_fact_orders.ctl',
DISCARD='/media/sf_VMSHARE/control-files/discard/fullstg_pkbc_fact_orders.txt',
LOG='/media/sf_VMSHARE/control-files/log/fullstg_pkbc_fact_orders.txt',
BAD='/media/sf_VMSHARE/control-files/bad/fullstg_pkbc_fact_orders.bad',
DATA='/media/sf_VMSHARE/fullstg_pkbc_fact_orders.csv' USERID=dws/dws@orcl

create or replace PROCEDURE  load_merge_fact_pkbc_orders IS
err_code NUMBER;
err_msg VARCHAR2(32000);
count_n number;
BEGIN
   MERGE INTO fact_pkbc_orders a
   USING stg_pkbc_fact_orders b
   ON (a.row_id = b.row_id)
 WHEN MATCHED THEN
   UPDATE SET
      a.order_id         = b.order_id                ,
      a.order_date       = b.order_date      ,
      a.is_returned      = b.is_returned     ,
```

```
    a.quantity         = b.quantity        ,
    a.discount         = b.discount        ,
    a.shipping_cost    = b.shipping_cost   ,
    a.profit           = b.profit          ,
    a.sales            = b.sales           ,
    a.ship_date        = b.ship_date       ,
    a.ship_mode        = b.ship_mode       ,
    a.address_id       = b.address_id      ,
    a.cust_id          = b.cust_id         ,
    a.sub_category_id  = b.sub_category_id ,
    a.product_id       = b.product_id      ,
                a.market               = b.market                    ,
    a.date_id          = b.date_id         ,
  a.TBL_LAST_DATE = to_timestamp(b.tbl_last_dt,'RRRR-MM-DD HH24:MI:SS')
WHEN NOT MATCHED THEN
  INSERT (
        row_id                  ,
        order_id          ,
        order_date        ,
        is_returned       ,
        quantity          ,
        discount          ,
        shipping_cost     ,
        profit            ,
        sales             ,
        ship_date         ,
        ship_mode         ,
        address_id        ,
        cust_id           ,
        sub_category_id   ,
        product_id        ,
        market                          ,
        date_id           ,
        TBL_LAST_DATE
```

```
        )
    VALUES (
        b.row_id                    ,
        b.order_id      ,
        b.order_date    ,
        b.is_returned   ,
        b.quantity      ,
        b.discount      ,
        b.shipping_cost ,
        b.profit        ,
        b.sales         ,
        b.ship_date     ,
        b.ship_mode     ,
        b.address_id    ,
        b.cust_id       ,
        b.sub_category_id ,
        b.product_id    ,
        b.market                    ,
        b.date_id       ,
    to_timestamp(b.tbl_last_dt,'RRRR-MM-DD HH24:MI:SS'));

    select count(*) into count_n from fact_pkbc_orders;
    dbms_output.put_line('count '||  count_n);
    commit;
EXCEPTION
   WHEN OTHERS THEN
      err_code := SQLCODE;
      err_msg  := SQLERRM;
      dbms_output.put_line('Error code ' || err_code || ': ' || err_msg);

END;
```