Team 3: Database Design

Simran Bawaskar, Saloni Bhutada, Aman Maheshwari,
Reha Patel, Niraj Sai Prasad, Sindhu Swaroop

DAMG 6210 Thursday Class

Table of Contents

Project 1

Team Members:

Name	Email ID	NUID
Simran Bawaskar	bawaskar.s@northeastern.edu	002957373
Saloni Bhutada	bhutada.s@northeastern.edu	002191872
Aman Maheshwari	maheshwari.am@northeastern.edu	001008819
Reha Patel (Admin)	patel.reh@northeastern.edu	002969991
Niraj Sai Prasad (Admin)	saiprasad.n@northeastern.edu	001006514
Sindhu Swaroop	swaroop.s@northeastern.edu	001006558

Project Topic: Order & Inventory Management for Electronic Items

Problem Statement: In the retail industry, it is vital that companies are able to keep up with the supply and demand on products they are offering. Some companies may choose to keep track of this supply and demand through physical books and paperwork, and over time this can result in misplaced orders by human error and overall customer dissatisfaction. Through this project we will demonstrate how a retail company that sells electronic items goes about digitally managing inventory based on orders that are fulfilled and returned .

Objectives:

- Customer management: In order for customers to place orders, they will need accounts and will thus need to be in the company database.
    - Adding an account: For new customers, they will need to create an account with information such as first name, last name, age, address, etc.
    - Deleting an account: For existing customers that are no longer interested in purchasing from the company or having accounts from the company they should be able to delete their account.
    - Updating an account: Existing customers may want to change their address or name and should be able to do so.
- Placing an order: Existing customers should be able to place an order with the company. Once they do this, they should get a confirmation or denial status if the product has gone out of stock. A confirmation status will result in the inventory changing.
- Cancelling an order: Customers can cancel the orders based on their flexibility.
    - Orders can be cancelled immediately after customers have placed the order and the product will return back to inventory.
    - If a customer tries to cancel an order after it has been shipped, the cancel request will be denied.

- Returning an order: Customers who want to return their orders should have an option to place a return request. Have a good dayOnce an item is returned, it will be added back to the inventory. However, returns must be placed within 15 days or the return request will be denied.
- Inventory check: Managing inventory and keeping track of the exact amount of stock is an important part of a retail business, in order for them to not run out of stock and hence improve delivery performance.
    - Adding inventory: Inventory could be added via return or via purchase from the company to restock.
    - Removing inventory: Inventory would be removed when it has been purchased by a customer.
- Product description: A simple product description table that includes the itemid, item description, item status (Available/Unavailable) [this is based on inventory count]
- Order status: Customers and backend business users can view order status based on which stage it is at.
    - Status Code: 1, Status: Draft Order
    - Status Code: 2, Status: Order Created
    - Status Code: 3, Status: Order Shipped
    - Status Code: 4, Status: Order Fulfilled
    - Status Code: 5, Status: Order Returned

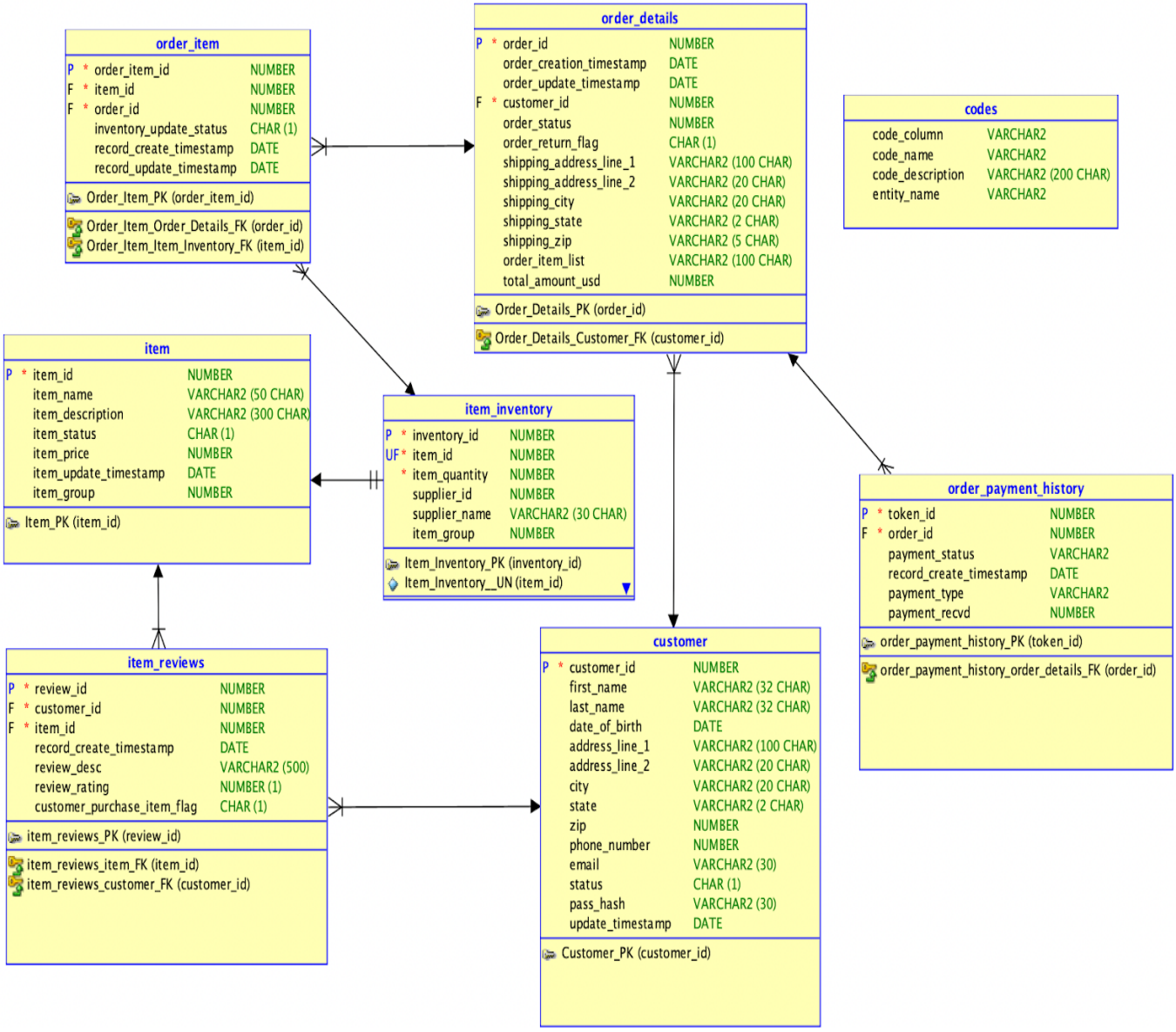
Project 2

Business Problem & Solution

In the retail industry, it is vital that companies are able to keep up with the supply and demand on products they are offering. Some companies may choose to keep track of this supply and demand through physical books and paperwork, and over time this can result in insufficient storage space, misplaced orders by human error, and overall customer dissatisfaction.

Our proposed solution demonstrates how a retail company that sells electronic items goes about digitally managing inventory based on orders that are fulfilled and returned. First and foremost, we allow customers to create accounts within our system that store data such as first name, last name, address, etc. This data is used when a customer creates an order with different items to ensure that the right orders are being associated with the right customers. In addition to this, when an order is placed, we will make sure that the correct amount of each item in the order is removed from the inventory. On the other hand, if a customer wants to return an item, we also have the capability to create a return order and readjust the inventory of the item.

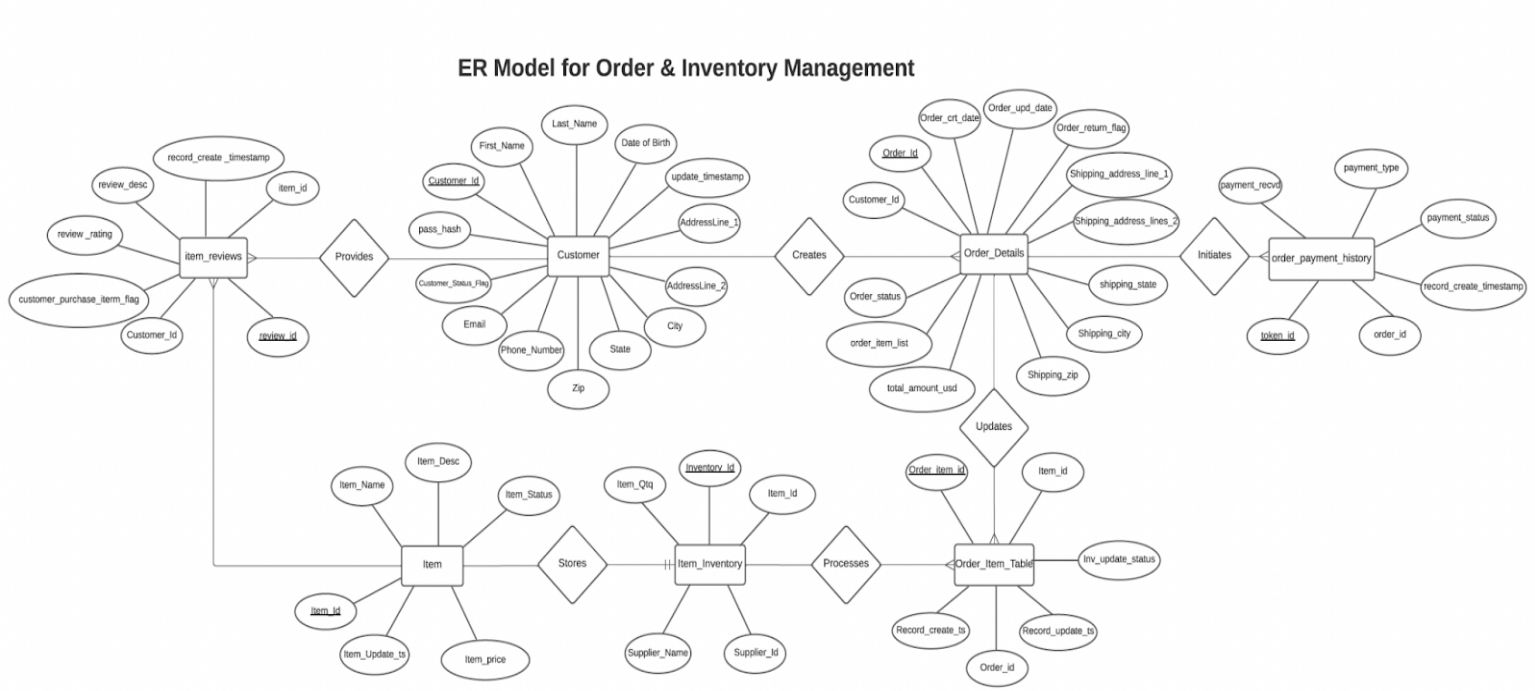
Database Design Document



Our database design consists of 8 tables:

- **Customer table** - Consists of customer demographics and login information
- **Order Details table** - Consists of specific details of orders made by the customers
- **Order Item table** - Consists of status of item inventory and relationships between orders and items
- **Item Inventory table** - Consists of item quantities
- **Item table** - Consists of item information
- **Item Reviews table -** Consists of all customer reviews of a particular table
- **Order Payment History table -** Consists of all payments paid against the order
- **Codes table -** A legend to all codes used in various tables in our dataset

Entity Relationship Diagram - First Draft



Entity and Attributes with Data Types

Item		
Attribute	**Datatype**	**Description**
item_id	NUMBER	This gives the ID of the item.
item-name	VARCHAR2	Used to specify name of the product
item_description	VARCHAR2	Used to give description of the product
item_status	CHAR(1)	This is a boolean which specifies if it is zero, it means item is out of stock and 1 means item is in stock.
item_price	NUMBER	Price of the product
item_update_timestamp	Date	This will be stored in date format
item_group	NUMBER	Stores which group the item belongs to

Item_Inventory		
Attribute	**Datatype**	**Description**
inventory_id	NUMBER	This is the primary key which specifies inventory ID
item_id	NUMBER	This gives the ID of the item.
item_quantity	NUMBER	This gives quantity of items required by the customer
supplier_id	NUMBER	This specifies ID of the supplier
supplier_name	VARCHAR2(30 CHAR)	This specifies ID of the supplier
item_group	NUMBER	Stores which group the item belongs to

Customer		
Attribute	**Datatype**	**Description**
customer_id	NUMBER	This specifies ID given to a customer. It is a primary key
first_name	VARCHAR2 (32 CHAR)	First name of customer
last_name	VARCHAR2 (32 CHAR)	Last name of customer
date_of_birth	DATE	This specifies date of birth of a customer.
address_line_1	VARCHAR2 (100 CHAR)	Address of the customer
address_line_2	VARCHAR2 (20 CHAR)	Address of the customer
city	VARCHAR2 (20 CHAR)	City in which customer lives
state	VARCHAR2 (2 CHAR)	State in which customer lives
zip	NUMBER	Zipcode of the address
phone_number	NUMBER	Phone number of the customer
email	VARCHAR2 (30)	Email ID of the customer
status	CHAR (1)	This specifies if the account holder is active or inactive
pass_hash	VARCHAR2(30)	Password is scrambled for security reasons.
update_timestamp	DATE	Give information regarding when was the account updated

Order_Item		
Attribute	**Datatype**	**Description**
order_item_id	NUMBER	This is the primary key of the table specifying
item_id	NUMBER	This given ID of an item
order_id	NUMBER	This given ID of an order
inventory_update_status	CHAR(1)	This gives status of inventory
record_create_timestamp	DATE	Information regarding when the record was created
record_update_timestamp	DATE	information regarding when the record was updated

Order_Details		
Attribute	**Datatype**	**Description**
order_id	NUMBER	This gives ID of a particular order. It is primary key of this table
order_creation_timestamp	DATE	This describes when the order was created
order_update_timestamp	DATE	This gives information about when order was updated
customer_id	NUMBER	This gives ID of a customer
order_status	NUMBER	This gives status of an order
order_return_flag	CHAR (1)	If the item is to be returned it will givw 1 value, if not it will give 0 value.
shipping_address_line_1	VARCHAR2 (100 CHAR)	Address of where the item should be shipped
shipping_address_line_2	VARCHAR2 (20 CHAR)	Address of where the item should be shipped
shipping_city	VARCHAR2 (20 CHAR)	City in which item should be shipped
shipping_state	VARCHAR2 (2 CHAR)	State in which item should be shipped
shipping_zip	VARCHAR2 (5 CHAR)	Zipcode of shipping address
order_item_list	VARCHAR2 (100 CHAR)	List of items that should be ordered
total_amount_usd	DOUBLE	Total amount for order

Codes		
Attribute	**Datatype**	**Description**
code_column	NUMBER	Column Name of Code
code_name	VARCHAR2	How the code looks like in the table
code_description	VARCHAR2	Description of the code.
entity_name	VARCHAR2	Table where code can be found

Item _Reviews		
Attribute	**Datatype**	**Description**
review_id *	NUMBER	A primary key that records the ID for each review
customer_id	NUMBER	ID of customer who made the review
item_id	NUMBER	ID of item that was reviewed
record_create_timestamp	DATE	Time at which review was created
record_update_timestamp	DATE	Time at which review was updated
review_desc	VARCHAR2	Review Description
review_rating	NUMBER	Rating of item from 1-5
customer_purchase_item_flag	BOOL	0 if customer did not purchase, 1 if customer purchased the item

Order_Payment_History		
Attribute	**Datatype**	**Description**
token_id *	NUMBER	A primary key that records the ID for each payment
order_id	NUMBER	Order ID for which payment was made
payment_status	VARCHAR2	Stores the status of payment (Failed/Processed/Incomplete)
record_create_timestamp	DATE	Time at which payment record was created
payment_type	VARCHAR2	Credit/Debit/Cheque
Payment_Recvd	NUMBER	Amount received in USD

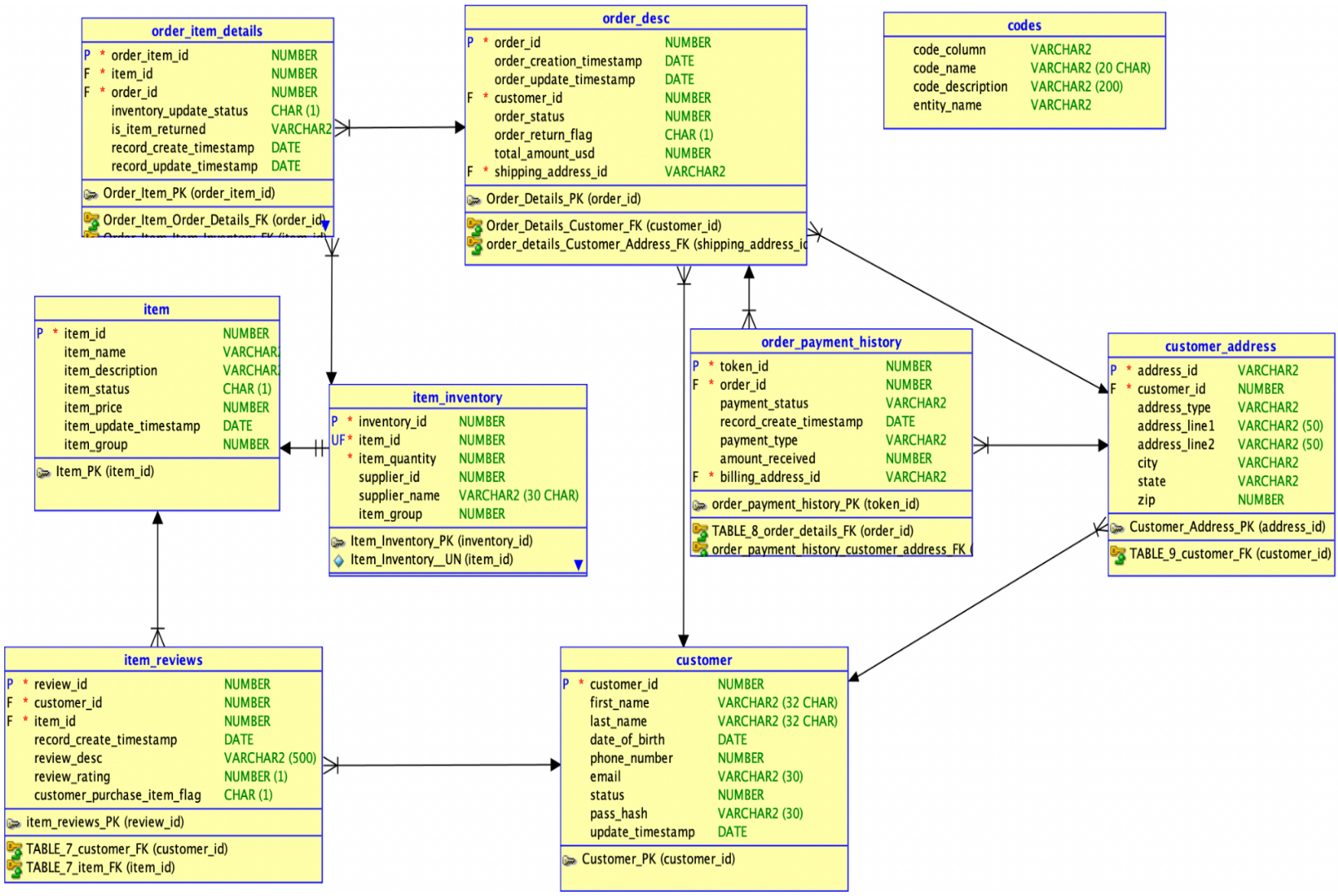
Project 3

1. **Business Problem & Solution**

In the retail industry, it is vital that companies are able to keep up with the supply and demand on products they are offering. Some companies may choose to keep track of this supply and demand through physical books and paperwork, and over time this can result in insufficient storage space, misplaced orders by human error, and overall customer dissatisfaction.

Our proposed solution demonstrates how a retail company that sells electronic items goes about digitally managing inventory based on orders that are fulfilled and returned. First and foremost, we allow customers to create accounts within our system that store data such as first name, last name, address, etc. This data is used when a customer creates an order with different items to ensure that the right orders are being associated with the right customers. In addition to this, when an order is placed, we will make sure that the correct amount of each item in the order is removed from the inventory. On the other hand, if a customer wants to return an item, we also have the capability to create a return order and readjust the inventory of the item.

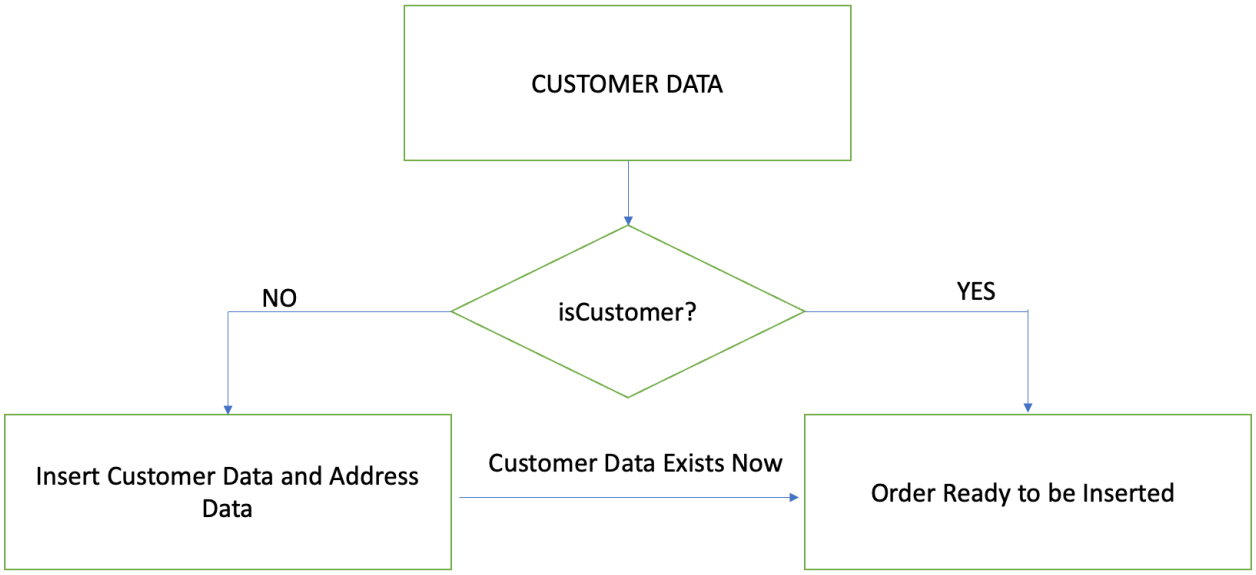
2. Database Model

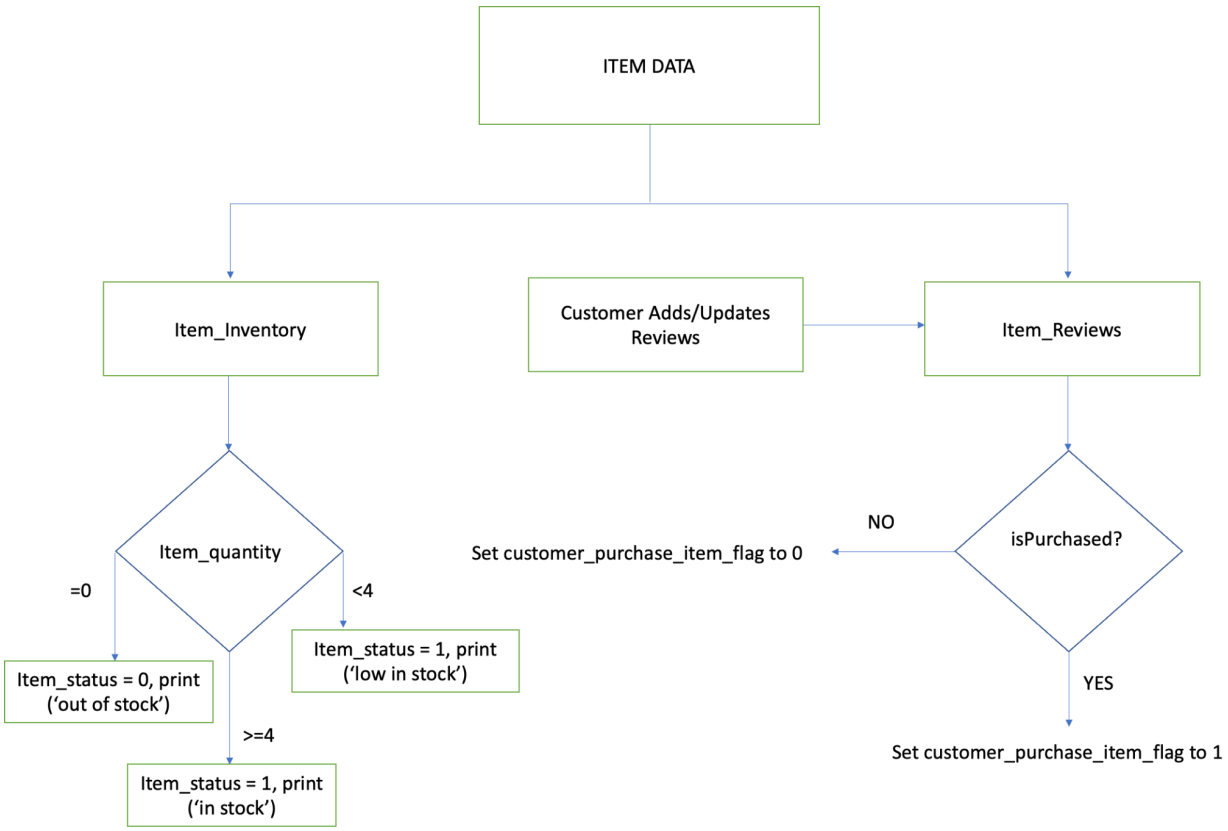


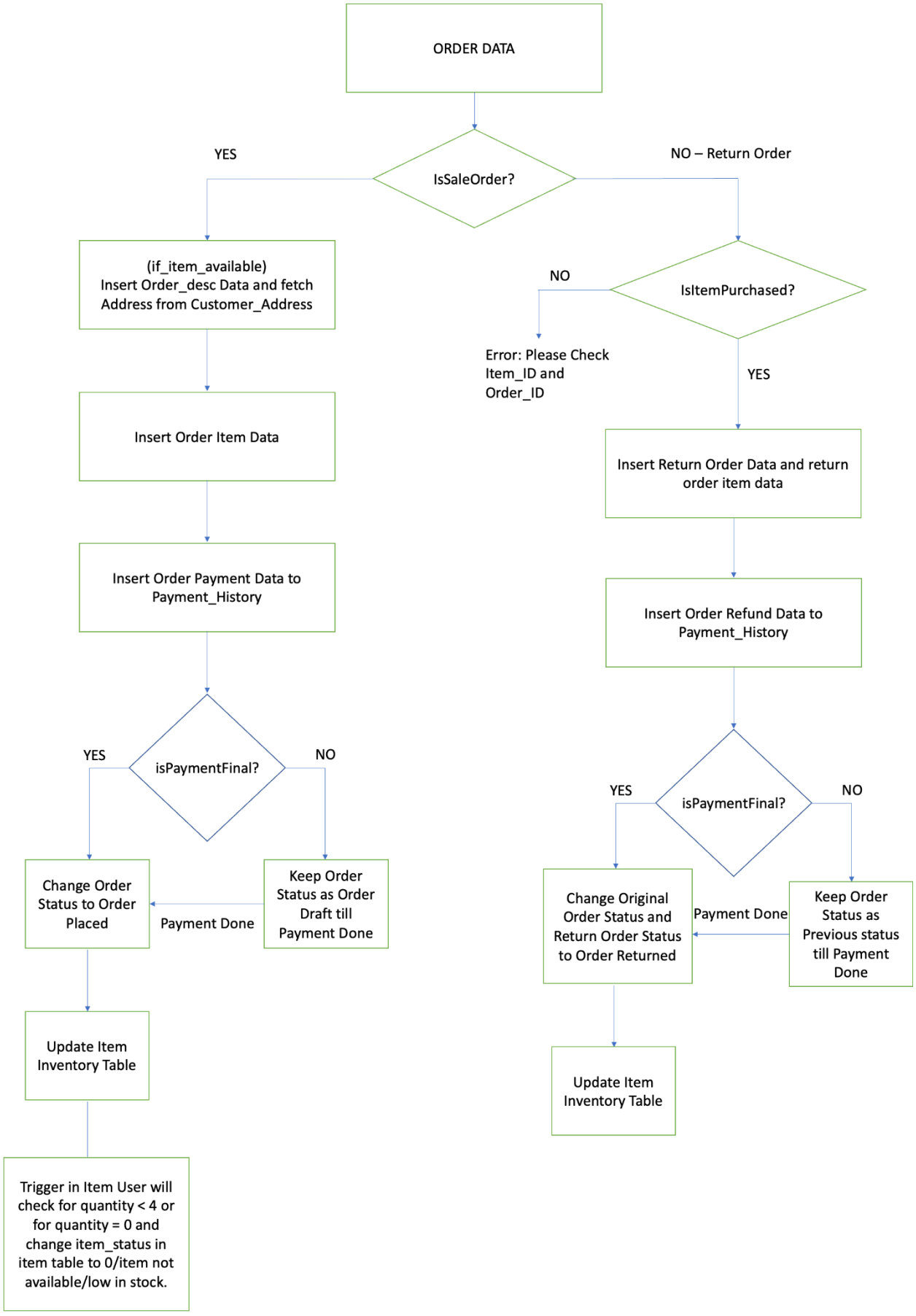
Our database design consists of 9 tables:

- **Customer table** - Consists of customer demographics and login information
- **Customer_Address table** - Consists of billing/shipping addresses
- **Order_Details table** - Consists of specific details of orders made by the customers
- **Order_Item table** - Consists of status of item inventory and relationships between orders and items
- **Item_Inventory table** - Consists of item quantities
- **Item table** - Consists of item information
- **Item_Reviews table -** Consists of all customer reviews of a particular table
- **Order_Payment_History table -** Consists of all payments paid against the order
- **Codes table -** A legend to all codes used in various tables in our database

3. Data Flow Diagrams






ORDER DATA
IsSaleOrder?
YES
NO – Return Order
(if_item_available)
Insert Order_desc Data and fetch Address from Customer_Address
Insert Order Item Data
Insert Order Payment Data to Payment_History
isPaymentFinal?
YES
NO
Change Order Status to Order Placed
Payment Done
Keep Order Status as Order Draft till Payment Done
Update Item Inventory Table
Trigger in Item User will check for quantity < 4 or for quantity = 0 and change item_status in item table to 0/item not available/low in stock.
IsItemPurchased?
NO
Error: Please Check Item_ID and Order_ID
YES
Insert Return Order Data and return order item data
Insert Order Refund Data to Payment_History
isPaymentFinal?
YES
NO
Change Original Order Status and Return Order Status to Order Returned
Payment Done
Keep Order Status as Previous status till Payment Done
Update Item Inventory Table

4. Business Rules

1) Our company wants to focus only on Electronic Items as a main product. Our inventory will include items like cell phones, cell phone accessories, laptops, etc.
2) In our project, shipping_address_id will point to the address_id column in the table Customer_Address. In the Customer_Address table, an address may be just a shipping address, just a billing address, or both. Each of these categories will be given a code accordingly.
3) Our company is relatively new and therefore we only have one supplier for each item at the moment. As the company continues to grow and expand, we may add more suppliers in the future.
4) If a customer wants to return an order then the return request will be considered as a new return order. The new return order will behave similar to a new order request, except the amount paid will be refunded and the item_quantity will be adjusted.
5) The Codes reference table will be pre-set for all the tables that have a status column.
6) A customer may deactivate their profile, but their tuple will not be removed from the customer table. Instead, their status will change from 1 (active) to 0 (inactive).
7) Our company does not accept cash payments - it only accepts debit, credit, and gift cards. We know a payment has been registered once the record is added to the Order_Payment_History table.
8) When a customer is trying to return an item purchased in a sale order, our system first checks if the customer has actually purchased the item in the said order. If everything returns TRUE, the order is return-eligible.

5. Views

Throughout our database we will be implementing views to facilitate our company's order management and fulfillment operations. The views below are integral for the process because we do not want to always display information such as update timestamps, statuses, passwords, etc. and views will help us decide which information is visible to which roles. The following are views we will be implementing:

- **Customer_View:** Consists of the following columns: customer.first_name, customer.last_name, customer_address.address_line1, customer_address.address_line2, customer_address.city, customer_address.zip, customer_address.state, customer.phone_number, customer.email will display's customers demographic information that can be accessed and modified by the customer.
- **Item_View:** Consists of the following columns: item.item_name, item.item_description, item.item_price, and item_inventory.item_quantity for customers to access the items and their details.
- **Order_Desc_Item_Payment_View:** This view will consist of three tables: Order_Item_Details, Order_Desc, and Order_Payment_history. It will consist of the following columns: order_desc.order_id, order_payment_history.payment_status, order_desc.order_status, order_desc.total_amount_usd, order_payment_history.payment_type, order_item_details.item_id to help the customers to keep track of every transaction.
- **Item_Reviews_View:** Consists of the following columns: item_reviews.item_id, item.item_name, item.item_description, item_reviews.review_desc, item_reviews.customer_id, customer.first_name, item_reviews.review_rating for customers to access the reviews of different items.
- **Order_Desc_Item_Customer_View:** This view will consist of three tables: Order_Item_Details, Order_Desc, and Customer. It will have the following columns: order_desc.order_id, order_desc.customer_id, customer.first_name, customer.last_name, order_desc.order_status, order_desc.total_amount_usd, and order_item_details.item_id, item.item_name.

6. Security Details

Team Member Roles

1. **Database Administrator - Simran:** The database administrator (DBA) will be responsible for monitoring and assisting the database in all stages from development to install. The DBA should be able to assist the team when the database connection lags or when the overall performance of the database is down. The DBA will have access to CREATE, UPDATE, and DELETE.
2. **Application DBA - Reha, Aman:** The application database administrator (DBA) will be responsible for creating the users within the database as well as determining which users have access to specific database objects and actions. Alongside this, the application DBA is responsible for granting roles and privileges when the developers need access. The Application DBA will have access to CREATE, UPDATE, and DELETE. Additionally, the Application DBA will be responsible for creating the Codes table which is referenced throughout the database.
3. **Monitoring User - Sindhu, Saloni, Niraj:** The monitoring user will be responsible for checking for performance issues within the database as well as performing metrics on the different procedures in the database. Such performance will help ensure that the database is performing well, and if there are any issues, the monitoring user should work with the DBA to fix the problem in a timely manner.
4. **Application Developer - All Members:** The application developers will break down the business requirements and propose possible solutions to the team. They will be utilizing PL/SQL to create tables, views, procedures, etc. that will satisfy the problem at hand in an efficient way. Alongside this, they are responsible for testing their code alongside the Application DBA to ensure that they have the appropriate roles required. Developers will be given privileges to CREATE and UPDATE tables, but they will need to work with a DBA to DELETE a table to ensure that no sensitive information is being deleted.

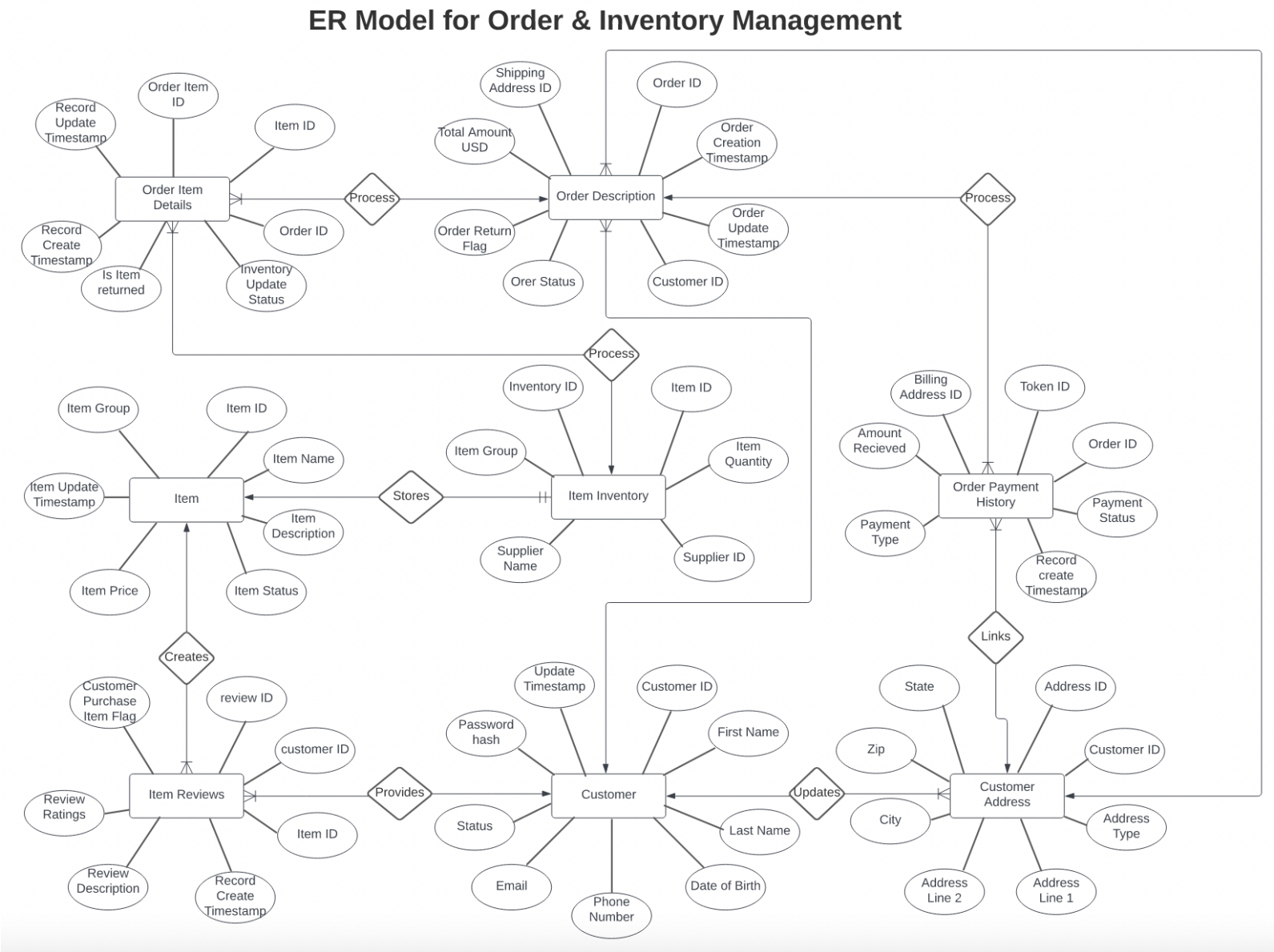
User Roles

1. **Customer_Owner:** The Customer_Owner role is responsible for creating and maintaining the tables called Customer and Customer_Address. They will be able to CREATE and UPDATE these tables and with the assistance of a DBA, they will also be able to DELETE. They will also have UPDATE privileges for Item_Reviews, Order_Payment_History, Order_Item_Details and Order_ Desc. Additionally, they will have READ access to Item_View, Order_Desc_Item_Payment_View, and Item_Reviews_View.
2. **Order_Owner:** The Order_Owner role is responsible for creating and maintaining the tables called Order_Desc, Order_Payment_History, and Order_Item Details. Similar to the previous role, they will be able to CREATE and UPDATE the previously mentioned tables and with DBA assistance they will be able to DELETE as well. They will also

have UPDATE privileges to Item_Inventory. Finally, they will have READ access to Item_View, Order_Desc_Item_Payment_View, and Customer_Address.

3. **Item_Owner:** The Item_Owner role is responsible for creating and maintaining the tables called Item, Item_Inventory, and Item_Reviews. Similar to the previous two roles, they will be able to CREATE and UPDATE the previously mentioned tables and with DBA assistance they will be able to DELETE as well. They will have READ access to Customer_View and Order_Desc_Item_Customer_View.

7. ER Diagram



Project 4

INSTRUCTIONS TO RUN OUR CODE:

-- Connect to your wallet and open the admin_code.sql and run

-- Create new connection for user item_owner with password: 'Itemowner2022'
-- Create new connection for user order_owner with password: 'Orderowner2022'
-- Create new connection for user customer_owner with password: 'Customerowner2022'

-- Open new worksheet for item_owner connection and run the code in file item_owner_create_table.sql

-- Open new worksheet for order_owner connection and run the code in file order_owner_create_table.sql

-- Open new worksheet for customer_owner connection and run the code in file customer_owner_create_table.sql

-- Exceptions that you may see upon re-running:

Admin_code.sql:

-- User may already exist, so we have handled this.

Owner sql files:

-- Table may already exist so we will not be creating a new table upon every re-run

-- If you are re-running multiple times, rerun the ENTIRE script altogether. Our script will delete all rows before inserting again so that multiple reruns do not fail.

-- Please run the reports SQL code in the respective user worksheets for validation if needed.

REPORTS:

We have done a few query validations to check if the inserts went through fine.

select count(*) as itemTableCount from item_owner.item;
select count(*) as inventoryTableCount from item_owner.item_inventory;
select count(*) as reviewTableCount from item_owner.item_reviews;

```
ITEMTABLECOUNT
--------------
           100


INVENTORYTABLECOUNT
-------------------
                100


REVIEWTABLECOUNT
----------------
             170
```

select * from item order by 1 desc;

	ITEM_ID	ITEM_NAME	ITEM_DESCRIPTION	ITEM_STATUS	ITEM_PRICE	ITEM_UPDATE_TIMEST...	ITEM_GROUP
1	100	iPhone 13 B...	Mobile	1	59.99	13-APR-22 11.52.00.000...	128
2	99	XPS 8920 To...	Accessories,Portab...	0	208.99	21-JUN-18 09.34.45.000...	120
3	98	Lenovo - 30...	Accessories,Portab...	0	121.99	03-JAN-18 05.30.13.000...	120
4	97	AudioQuest ...	Accessories,Portab...	1	124.19	13-JUN-18 07.39.21.000...	120
5	96	PELICAN - P...	Computers,Hard Dri...	1	137.99	13-JUN-18 07.37.55.000...	112
6	95	Toshiba - 2...	Computers,Hard Dri...	1	159.09	13-JUN-18 07.51.04.000...	112

We performed the check for all our other tables and everything tested good.

We did some exploratory data analysis with a few business questions for our reports:

- List out all the return orders created over all time:

select count(*) from order_desc where order_return_flag = 1;

COUNT(*)
5

- Show the different product groups and the average price for each group:

select unique(item_group),round(AVG(item_price), 2) as avg_prc from item GROUP by item_group order by item_group;

	ITEM_GROUP	AVG_PRC
1	1	104.99
2	8	73.41
3	16	267.99
4	24	164.98
5	32	389.79
6	40	492.59
7	48	167.16
8	56	156.97
9	64	102.25
10	72	124.97
11	80	83.55
12	88	165.97
13	96	43.74
14	104	87.83
15	112	156.56
16	120	151.72
17	128	59.99

- Find all the inactive customers using our application

select customer_id, status, first_name from customer where status <> 1 and first_name is not null order by customer_id;

We have 10 customers who are inactive.

	CUSTOMER_ID	STATUS	FIRST_NAME
1	9	0	Sage
2	19	0	Yuki
3	20	0	Fletcher
4	26	0	Allene
5	61	0	Blondell
6	64	0	Carmelina
7	79	0	Viva
8	80	0	Elza
9	92	0	Cory
10	93	0	Danica

- Display count of products who's inventory stock is less than 10

select count(*) from item_inventory where item_quantity < 10;

COUNT(*)
14

- Display count of all item reviews that are not placed by a customer.

select count(*) from item_reviews where customer_purchase_item_flag = 0;

There are 81 non-customer reviews [maybe bots or customers who did not make purchase]

COUNT(*)
81